JOHNSON GRANT
IN-61-CR
96744

Advanced Software Development Workstation Engineering Scripting Language Graphical Editor DRAFT Code Documentation

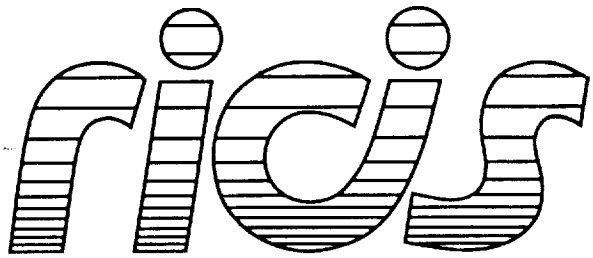
P. 90

Inference Corporation

8/30/91

Cooperative Agreement NCC 9-16
Research Activity No. SE.41

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division



Research Institute for Computing and Information Systems
University of Houston-Clear Lake

INTERIM REPORT

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Inference Corporation. Dr. Anthony Lekkos, Associate Professor, Computer and Information Sciences, served as RICIS research coordinator.

Funding was provided by the Information Technology Division, Information Systems Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this activity was Robert Savely of the Information Technology Division, Information Systems Directorate, NASA/JSC.

Inference Corporation
Owner ralston (Elizabeth Ralston)
Host Quaestor
printer imagen
Date Fri Aug 30 09:13:57 1991

```
c12 0
c11 0
c10 0
c9 0
c8 0
c7 0
c6 0
c5 0
c4 0
c3 0
c2 0
c1 1
printwheel sd_cour12
rules no
outlines no
formlength 60
formwidth 80
leftmargin 0
at (0.4 0.25 cc)
window (7.5 10.5 0)
imagespace (0 960 -8 520)
imagesize (7.5 10.5)
jobheader on
pagecollation on
jamresistance on
language printer
```

```
System Version TURBO UltraScript 6.0T Rev. A IP/II, Serial #90:12:48
Page images processed: 88
Pages printed: 88

Paper size (width, height):
  2560, 3328
Document length:
  136249 bytes
```

```
;;  esl-screens.art


;(DEFSCHEMA PANEL-SPEC
;  (DISPATCH-TABLE)
;  (EXPOSED)
;  (HAS-ITEM-SPECS)
;  (PANEL-ID)
;  (PANEL-NAME)
;  (TARGET-POINTER)
;  (VIEW-POINTER)
;  (DEFAULT-BACKGROUND-COLOR)
;  (NON-DEFAULT-BACKGROUND-COLOR))

(defschema notes-panel
        (is-a panel-spec)
        (currently-displayed-component)
)

(defschema component-notes-panel
        (instance-of notes-panel)
)

(defschema node-notes-panel
        (instance-of panel-spec)
        (currently-displayed-node)      ;has node
)

(defschema select-constant-value-panel
        (is-a modal-panel)
        (has-parent-panel)      ;pointer to constant-to-node-panel
)

(defschema connector-details-panel
        (is-a panel-spec)
        (currently-displayed-connector-group)   ;value is connector-group
                                                ;schema
        (connector-reps)                ;value is pointer to an array
                                        ; of pointers to C
                                        ; connector_rep structs -
                                        ; see file connector_rep.h

        (connector-reps-size)           ;size of the array pointed to
                                        ; by connector-reps slot
)

(defschema node-to-node-connector-details-panel
        (instance-of connector-details-panel)
        (source-node)                   ;value is node schema
        (destination-node)              ;value is node schema

        (input-port-reps)               ;value is pointer to a NULL
                                        ; terminated array of pointers to
                                        ; C port_rep structs

        (output-port-reps)              ;like input-port-reps
```

```
)

(defschema constants-to-node-connector-details-panel
        (instance-of connector-details-panel)
        (destination-node)
        (output-port-reps)
)

(defschema graph-port-to-node-connector-details-panel
        (instance-of connector-details-panel)
        (destination-node)
        (output-port-reps)
)

(defschema node-to-graph-port-connector-details-panel
        (instance-of connector-details-panel)
        (source-node)
        (input-port-reps)
)


(defschema node-details-panel
        (instance-of panel-spec)
        (for-node)
)

(defschema component-details-panel
        (instance-of panel-spec)
        (for-component)
)

(defschema graph-port-details-panel
        (instance-of panel-spec)
        (for-graph)
        (input?)
)


(defschema dummy-esl-panel
        (instance-of panel-spec)
        (view-path)             ;; sequence, always begins with root-graph
                                ;;    and ends with current-graph
        (edit-object)
        (connector-source)
        (connector-destination)
)
/*
**
**  file: esl_panels.c
**
**  Event handling functions for the ESL panels.
**
*/


#include "esl_panels.h"
```

```
#include "access_protos.h"
#include "artsymbols.h"
#include "defs.h"
#include "connector_rep.h"
#include "string_utils.h"
#include "type_check.h"
#include "esl_editor_panel.h"

char *strdup(/* char *s */);



/*****************************************************************************/

#ifdef EXTRA_FUNCTIONS_H

void determine_CD_panel_type (/* art_symbol panel_schema,
                                 boolean *node_to,
                                 boolean *constant_to,
                                 boolean *graph_port_to,
                                 boolean *to_node,
                                 boolean *to_graph_port */);
#endif EXTRA_FUNCTIONS_H

/*
**      void determine_CD_panel_type (art_symbol panel_schema,
**                                    boolean *node_to,
**                                    boolean *constant_to,
**                                    boolean *graph_port_to,
**                                    boolean *to_node,
**                                    boolean *to_graph_port);
**
**  Determines the panel type of a connector details panel.
**
**  Where
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      node_to, constant_to, graph_port_to, to_node, to_graph_port
**  are pointers to booleans that are set by this function.  Only one of
**  the "<x>_to" variables will be set TRUE and only one of the "to_<x>"
**  variables will be set TRUE.  All others will be set FALSE
**
*/

void determine_CD_panel_type (panel_schema,
                              node_to, constant_to, graph_port_to,
                              to_node, to_graph_port)
art_symbol panel_schema;
boolean *node_to, *constant_to, *graph_port_to;
boolean *to_node, *to_graph_port;
{
        *node_to = *constant_to = *graph_port_to = *to_node = *to_graph_port
```

```
                = 0L;

        if (a_eq(panel_schema, NODE_TO_NODE_CONNECTOR_DETAILS_PANEL))
                *node_to = *to_node = 1L;

        else if (a_eq(panel_schema,
                        CONSTANTS_TO_NODE_CONNECTOR_DETAILS_PANEL))
                *constant_to = *to_node = 1L;

        else if (a_eq(panel_schema,
                        GRAPH_PORT_TO_NODE_CONNECTOR_DETAILS_PANEL))
                *graph_port_to = *to_node = 1L;

        else /* (a_eq(panel_schema,
                        NODE_TO_GRAPH_PORT_CONNECTOR_DETAILS_PANEL)) */
                *node_to = *to_graph_port = 1L;
}




/*****************************************************************************
******************************************************************************
**
**   Connector Details Panels
**
*/

/*
**  Common elements:
*/



/*
**  EVENT_HANDLER CD_Selected_Connector (art_symbol panel_schema,
**                                       char *value);
**
**
**  This function handles the "item has been selected" event for the
**  Connectors textlist.  All four types of Connector Details Panels have
**  this field.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      value is the string that was selected on the Connectors textlist
**
********
**
**  In all cases, the Notes field is populated with the notes for the
```

```
**  selected connector.
**
**  If the panel is a "Node To" CD Panel, then the corresponding source
**  port item in the Souces textlist is selected.
**
**  If the panel is a "To Node" CD Panel, then the corresponding
**  destination port item in the Destinations textlist is selected.
**
**  If the panel is a "Constant to" CD Panel, then the Constant_Value text
**  field is populated with the source of the connector item selected.
**
**  If the panel is a "Graph Port" CD Panel, then the Graph_Port_Name
**  field is populated with the source of the selected connector item.
**
********
**
**  The source and destination of the selected connector menu item are
**  extracted with extract_connector_source() and
**  extract_connector_destination().  The strings returned by these
**  functions are freed here.
**
**  The value returned by connector_menu_item_[source|destination] is
**  suitable for the Sources textlist (Node to panels), the Destination
**  textlist (to Node panels), and Constant Value field (Constant to
**  panels), but not for the Graph Port Name field.  For this field it
**  must have the type information stripped off.  This is done with
**  extract_port_name().  The string returned by this function must
**  be freed here.
**
**  The associated notes are found by using search_for_connector_rep() to
**  get a pointer to the connector_rep structure.  The notes field
**  contains the notes.
**
**  It is an internal error for there to be no connector_rep associated
**  with the selected connector menu item (if search_for_connector_rep()
**  returns NULL.  The best behavior for this case is to do nothing.
**
*/

EVENT_HANDLER CD_Selected_Connector (panel_schema, value)
art_symbol panel_schema;
char *value;
{
        connector_rep *this_connector_rep;
        char *source, *destination;
        char *notes;
        char *graph_port_name;
        boolean node_to, constant_to, graph_port_to, to_node, to_graph_port;
        char warning_message[LONG_STRING_LENGTH];


        if (!strcmp(value, NO_CONNECTORS_VALUE))
                return;

        this_connector_rep = search_for_connector_rep(panel_schema, value);
        if (this_connector_rep)
```

```
        {
                if (this_connector_rep->notes)
                        update_text_field(panel_schema, NOTES_FIELD,
                                          this_connector_rep->notes);
                else
                        update_text_field(panel_schema, NOTES_FIELD, "");
        }
        else
        {
                sprintf(warning_message,
"WARNING: Cannot find connector_rep for menu-item = %s\n",
                        value);
                display_warning_message(panel_schema, warning_message);
                return;
        }


        determine_CD_panel_type (panel_schema,
                                 &node_to, &constant_to, &graph_port_to,
                                 &to_node, &to_graph_port);

        source = extract_connector_source (value);
        destination = extract_connector_destination (value);

        if (node_to)
                select_item_in_textlist(panel_schema, SOURCES_FIELD, source);
        else if (constant_to)
                update_text_field(panel_schema, CONSTANT_VALUE_FIELD, source);
        else /* (graph_port_to) */
        {
                graph_port_name = extract_port_name(source);
                update_text_field(panel_schema, GRAPH_PORT_NAME_FIELD,
                                  graph_port_name);
                rtn_memory(graph_port_name);
        }

        if (to_node)
                select_item_in_textlist(panel_schema, DESTINATIONS_FIELD,
                                        destination);
        else /* (to_graph_port) */
        {
                graph_port_name = extract_port_name(destination);
                update_text_field(panel_schema, GRAPH_PORT_NAME_FIELD,
                                  graph_port_name);
                rtn_memory(graph_port_name);
        }

        rtn_memory(source);
        rtn_memory(destination);
}



/*
**  EVENT_HANDLER CD_Changed_Notes (art_symbol panel_schema,
```

```
**                                      char *value);
**
**  This function handles the "text field has been changed and exited" event
**  for the Notes text field.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      value is the string that has been entered in the Notes field
**
********
**
**  If an item is selected in the Connectors textlist, then the new value
**  of the notes must be associated with that connector.
**
**  If no item is selected in the Connectors textlist, then no action is
**  needed.
**
**
**  COMMENTS:
**
**  * Mechanism needed to associate connectors with notes.
**
********
**
**  The selected connectors textlist item is found with
**  get_value_from_field().  The connector_rep structure is found with
**  search_for_connector_rep().  The returned connector_rep's notes member
**  contains the notes for that connector item.  The old notes are freed
**  and a pointer to the new notes is stored in the notes member.
**
**  It is an internal error for there to be no connector_rep structure
**  associated with any item in the connectors textlist.  In this case,
**  search_for_connnector_rep() will return NULL. The best behavior for
**  this case is to do nothing.
**
*/

EVENT_HANDLER CD_Changed_Notes (panel_schema, value)
art_symbol panel_schema;
char *value;
{
        char *connector_menu_item;
        connector_rep *this_connector_rep;
        char *old_notes;

        connector_menu_item = get_selected_item_from_textlist
                (panel_schema, CONNECTORS_FIELD);

        if (connector_menu_item)
        {
                this_connector_rep =
                        search_for_connector_rep(panel_schema,
                                                 connector_menu_item);
```

```
                if (this_connector_rep)
                {
                        old_notes = this_connector_rep->notes;
                        if (old_notes)
                                rtn_memory(old_notes);
                        this_connector_rep->notes = strdup(value);
                }
        }
}



/*
**  EVENT_HANDLER CD_Selected_Connect (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Connect button.  Normally this causes a new connector object to be
**  defined.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**  1. The connection should satisfy some validity checking.
**
**  Since data type is the only thing that can be checked here and it is
**  checked when items are selected in the Sources and Destinations
**  textlists (or Constant Value field or Graph Port field), all that need

**  be done is verify that both the "source" field and "destination" field
**  have values.
**
**    If this is not the case, a warning panel should be popped up
**  explaining the problem.  No connector is defined.
**
**  2. The connector is now defined and the contents of the notes field
**  associated with the new connector.
**
**  3. The Connections textlist is populated with a new item for the new
**  connector definition.  The new item is selected.
**
**
**  COMMENTS:
**
**  * Mechanism needed to define and undefine connectors.
**
********
**
**  If a value is already selected on the connectors textlist, do nothing.
**
**  Type checking is done by seeing that the source and destination fields
**  have values.  (Depending on the panel type, the source is either the
```

```
**  selected item on the Sources textlist, the value of the Constant Value
**  field, or the value of the Graph Port Name field.  The Destination is
**  either the selected item on the Destinations textlist or the value of
**  the Graph Port Name field.
**
**
**  If the source or destination are a Graph Port, the type information is
**  added first to the port name with make_port_type_menu_item and the
**  type of the connecting node port source or destination.  Then, the new
**  connector menu item is constructed with make_connector_menu_item().
**  The type name is not added to the Constant Value Field value on
**  Constant to Node panels.
**
**
**  The function search_for_connector_rep() is used to see if this
**  connector has been previously defined for this panel.  If so, then
**  re_associate_connector() is called to reassociate the connector item
**  and the memory returned by make_connector_menu_item is freed()
**
**  If not, the function add_new_connector_rep() is used to create the new
**  connector_rep structure.
**
**  The connectors textlist is repopulated by populate_connectors_textlist().
**
**  Finally the new connector item is selected in the connectors textlist
**  with select_item_in_textlist().
**
*/

EVENT_HANDLER CD_Selected_Connect (panel_schema)
art_symbol panel_schema;
{
        char *connector_menu_item;
        char *source, *destination, *graph_port_name;
        char *other_type;
        connector_rep *this_connector_rep;
        char *notes;
        boolean node_to, constant_to, graph_port_to, to_node, to_graph_port;
        art_symbol connector_group, source_schema, destination_schema;

        connector_menu_item =
                get_selected_item_from_textlist(panel_schema,
                                                CONNECTORS_FIELD);
        if (connector_menu_item && strcmp(connector_menu_item, ""))
                return;

        determine_CD_panel_type (panel_schema,
                                 &node_to, &constant_to, &graph_port_to,
                                 &to_node, &to_graph_port);

        if (node_to)
                source = get_selected_item_from_textlist (panel_schema,
                                                          SOURCES_FIELD);
        else if (constant_to)
                source = get_value_from_field (panel_schema,
                                               CONSTANT_VALUE_FIELD);
```

```
        else /* (graph_port_to) */
                source = get_value_from_field (panel_schema,
                                               GRAPH_PORT_NAME_FIELD);

        if (to_node)
                destination =
                        get_selected_item_from_textlist (panel_schema,
                                                         DESTINATIONS_FIELD);
        else /* (to_graph_port) */
                destination = get_value_from_field (panel_schema,
                                                    GRAPH_PORT_NAME_FIELD);

        if (!source || !destination ||
            !strcmp(source, "") || !strcmp(destination, ""))
        {
                display_warning_message(panel_schema,
"ERROR: Must select the source and destination of the connector before hitting C
                return;
        }

        if (graph_port_to)
        {
                other_type = extract_type_name(destination);

                graph_port_name = source;
                source = make_port_type_menu_item (source, other_type);

                rtn_memory(other_type);
        }
        else if (to_graph_port)
        {
                other_type = extract_type_name(source);

                graph_port_name = destination;
                source = make_port_type_menu_item (destination, other_type);

                rtn_memory(other_type);
        }

        connector_menu_item = make_connector_menu_item(source, destination);

        if (graph_port_to)
                rtn_memory(source);
        else if (to_graph_port)
                rtn_memory(destination);


        notes = get_value_from_field (panel_schema, NOTES_FIELD);
        this_connector_rep = search_for_connector_rep(panel_schema, connector_me
        if (visible_connector_p (this_connector_rep))
        {
                re_associate_connector (this_connector_rep, notes);
                rtn_memory(connector_menu_item);
        }
        else
                add_new_connector_rep (panel_schema,
```

```
                                        connector_menu_item,
                                        NULL,
                                        notes);


        /* repopulate... */
        connector_group = a_get_schema_value(panel_schema,
                                        CURRENTLY_DISPLAYED_CONNECTOR_GROUP);
        source_schema = a_get_schema_value (connector_group, SOURCE_NODE);
        destination_schema =
                a_get_schema_value (connector_group, DESTINATION_NODE);
        populate_connectors_textlist (panel_schema,
                                      source_schema, destination_schema);

        select_item_in_textlist(panel_schema, CONNECTORS_FIELD,
                                connector_menu_item);
}



/*
**  EVENT_HANDLER CD_Selected_Disconnect (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Disconnect button for the Node_To_Node Connector Details Panel.
**  Normally this causes a defined connector object to become undefined.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**  This event is valid if an item is selected on the Connectors textlist.
**  If this is not the case, then a warning panel is popped up and no
**  connector is undefined.
**
**  Otherwise, the selected connector is undefined.  Items are unselected
**  in the Connectors, Sources, and Destinations textlist (or constant
**  value field is populated with an empty string, or graph port name
**  field is populated with an empty string).  The Notes field is
**  populated with an empty string.
**
**  COMMENTS:
**
**  * Mechanism needed to define and undefine connectors.
**
********
**
**  The function search_for_connector_rep() is used to find the
**  connector_rep associated with the selected item in the connectors
**  textlist.  The connector is disassociated with
**  disassociate_connector_rep() and the connectors textlist is
**  repopulated by populate_connectors_textlist().
**
```

```
**  The source and destination fields are "cleared."  In the case of the
**  Sources textlist and Destinations textlist, the selection is cleared.
**  In the case of the Constant Value text field or the Graph Port Name
**  text field, the fields are populated with the empty string.
**
**  It is an internal error for there to be no connector_rep structure
**  associated with any item in the connectors textlist.  In this case,
**  search_for_connnector_rep() will return NULL. The best behavior for
**  this case is to do nothing.
**
*/

EVENT_HANDLER CD_Selected_Disconnect (panel_schema)
art_symbol panel_schema;
{
        char *connector_menu_item;
        connector_rep *this_connector_rep;
        art_symbol connector_group, source_schema, destination_schema;
        boolean node_to, constant_to, graph_port_to, to_node, to_graph_port;

        connector_menu_item =
                get_selected_item_from_textlist (panel_schema,
                                                 CONNECTORS_FIELD);
        if (!connector_menu_item || !strcmp(connector_menu_item, ""))
                return;

        this_connector_rep = search_for_connector_rep (panel_schema,
                                                       connector_menu_item);
        if (visible_connector_p (this_connector_rep))
        {
                disassociate_connector_rep (panel_schema,
                                            this_connector_rep);

                /* repopulate... */
                connector_group = a_get_schema_value(panel_schema,
                                        CURRENTLY_DISPLAYED_CONNECTOR_GROUP);
                source_schema = a_get_schema_value (connector_group,
                                                    SOURCE_NODE);
                destination_schema =
                        a_get_schema_value (connector_group,
                                            DESTINATION_NODE);
                populate_connectors_textlist (panel_schema,
                                              source_schema,
                                              destination_schema);

                determine_CD_panel_type (panel_schema,
                                         &node_to, &constant_to,
                                         &graph_port_to,
                                         &to_node, &to_graph_port);
                if (node_to)
                        unselect_items_on_textlist(panel_schema,
                                                   SOURCES_FIELD);
                else if (constant_to)
                        update_text_field(panel_schema,
                                          CONSTANT_VALUE_FIELD, "");
                else /* (graph_port_to) */
```

```
                        update_text_field(panel_schema,
                                          GRAPH_PORT_NAME_FIELD, "");

                if (to_node)
                        unselect_items_on_textlist(panel_schema,
                                                   DESTINATIONS_FIELD);
                else /* (to_graph_port) */
                        update_text_field(panel_schema,
                                          GRAPH_PORT_NAME_FIELD, "");

                update_text_field (panel_schema, NOTES_FIELD, "");
        }
}



/*
**  EVENT_HANDLER CD_Selected_Ok (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the Ok
**  button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**  It has the effect of an Apply followed by a Close.
**
**  All newly defined connectors must be created and all freshly deleted
**  connectors can now be destroyed.
**
**  After the connectors have been changed, constraints must be allowed to
**  fire.
**
**  If the Apply causes constraints to be violated the panel should not be
**  closed.  If no constraints are violated, the panel is dismissed.
**
**
**  COMMENTS:
**
**  * Mechanism needed to define and undefine connectors.
**
**  * Mechanism needed to apply constraints.
**
********
**
**  Connectors are created and destroyed with
**  create_and_destroy_connectors().  Constraints are propogated with
**  propogate_constraints().  If no constraints are violated, CD_Close()
**  is called to dismiss the panel.
**
*/
```

```
EVENT_HANDLER CD_Selected_Ok (panel_schema)
art_symbol panel_schema;
{
        create_and_destroy_changed_connectors (panel_schema);
        if (propogate_constraints())
                CD_Selected_Close (panel_schema);
}



/*
**  EVENT_HANDLER CD_Selected_Apply (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Apply button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**  All newly made connectors must be created and all freshly deleted
**  connectors can now be destroyed.
**
**  After the connectors have been changed, constraints must be allowed to
**  fire.
**
**
**  COMMENTS:
**
**  - Mechanism needed to define and undefine connectors.
**
**  - Mechanism needed to apply constraints.
**
********
**
**  Connectors are created and destroyed with
**  create_and_destroy_connectors().  Constraints are propogated with
**  propogate_constraints().
**
*/

EVENT_HANDLER CD_Selected_Apply (panel_schema)
art_symbol panel_schema;
{
        create_and_destroy_changed_connectors (panel_schema);
        propogate_constraints();
}


/*
```

```
**  EVENT_HANDLER CD_Selected_Close (panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Close button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**  Dismisses the Connector_Details_Panel.
**
**  Memory used by the connector_rep structures is freed with
**  free_up_reps().  The panel is dismissed with
**  dispose_of_screen().
*/

EVENT_HANDLER CD_Selected_Close (panel_schema)
art_symbol panel_schema;
{
        free_up_reps(panel_schema);
        dispose_of_screen(panel_schema);
}



/*
**  Node To, To Node features
*/


/*
**  EVENT_HANDLER CD_Selected_Source (art_symbol panel_schema,
**                                    char *value);
**
**  EVENT_HANDLER CD_Selected_Destination (art_symbol panel_schema,
**                                         char *value);
**
**
**  These functions handle the "item has been selected" event for the
**  Sources textlist and the Destinations textlist fields respectively on
**  the Node_To_Node Connector Details Panel.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      value is the string that was selected on the Sources
**              or Destinations textlist
**
********
```

```
**
**  First, if an item is selected on the Connections textlist, then it is
**  unselected and the Notes Field is cleared.
**
**  For the case of CD_Selected_Source():
**
**      This function then parses the menu item to determine the type
**  of port which has been selected.  If the panel is a To Node panel,
**  then it retrieves any selected menu item from the Destinations
**  textlist and parses that to determine the type.  If the types are not
**  compatible or if there is already a connector defined with that
**  destination and a different source (this would correspond to an
**  unallowable merge), then any the item selected on the Destinations
**  Textlist is unselected.
**
**      If the above results in no item being selected on the
**  Destinations textlist, then a check is made as to whether the selected
**  source is the source for a defined connection.  If so, that connection
**  is selected and the Destinations textlist and Notes field populated
**  accordingly.
**
**      If an item is still selected on the Destinations textlist,
**  then a check is made as to whether the Source/Destination pair
**  corresponds to an item already on the Connections textlist.  If so,
**  that item is selected and the panel repopulated.
**
**      If the panel is a To Graph Port panel, then ...
**
**  For the case of CD_Selected_Destination():
**
**      (ed: must explain how the above is changed.  Also must explain
**  about the Constant Value field)
**
********
**
**  The data type of menu items are extracted with extract_type_name().
**  The returned string is freed here.
**
**  An existing connector object is searched for by either
**  search_for_connector_rep_by_source() or
**  search_for_connector_rep_by_destination().  The returned connector_rep
**  structure will have the connector menu item to be selected and the
**  corresponding notes.
**
**  To check types of two node ports, the function type_check_two_ports()
**  is used.  (ed: this function is undefined as of yet.)  For graph port
**  source or destination, data type checking is done with
**  type_check_graph_port().  For constant source, data type checking is
**  done with type_check_constant().
**
**
**
**  COMMENTS:
**
**  - The type_check_constant() and type_check_graph_port() currently
**  accept strings for data types, but I think they'll really want a
```

```
**  DATA_TYPE schema.  The PORT schema contains a PORT_DATA_TYPE slot that
**  points to the DATA_TYPE schema for the PORT.  But getting to the PORT
**  schema from the the port name is difficult.
**
**  - A port_rep structure is needed when defining graph ports.  Will be
**  defined later.
**
*/

EVENT_HANDLER CD_Selected_Source (panel_schema, value)
art_symbol panel_schema;
char *value;
{
        char *selected_type;
        char *destination_menu_item;
        char *destination_type;
        char *destination_of_source_connector;
        boolean to_node, to_graph_port;
        connector_rep *conn_rep;

        if (!strcmp(value, NO_PORTS_VALUE))
                return;

        to_node = a_eq(panel_schema, NODE_TO_NODE_CONNECTOR_DETAILS_PANEL);
        to_graph_port = a_eq(panel_schema,
                             NODE_TO_GRAPH_PORT_CONNECTOR_DETAILS_PANEL);

        unselect_items_on_textlist (panel_schema, CONNECTORS_FIELD);

        selected_type = extract_type_name(value);
        if (to_node)
        {
                destination_menu_item =
                        get_selected_item_from_textlist (panel_schema,
                                                         DESTINATIONS_FIELD);
                if (destination_menu_item &&
                    strcmp(destination_menu_item, ""))
                {
                        destination_type =
                                extract_type_name(destination_menu_item);

                        conn_rep = search_for_connector_rep_by_destination
                                (panel_schema, destination_menu_item);

                        if (visible_connector_p (conn_rep) ||
                            !compare_data_types(selected_type,
                                                destination_type))
                        {
                                unselect_items_on_textlist(panel_schema,
                                                           DESTINATIONS_FIELD);
                                destination_menu_item = NULL;
                        }

                        rtn_memory(destination_type);
                }
```

```
if (!destination_menu_item)
{
        conn_rep = search_for_connector_rep_by_source
                (panel_schema, value);
        if (visible_connector_p (conn_rep))
        {
                select_item_in_textlist(panel_schema,
                                        CONNECTORS_FIELD,
                                        conn_rep->menu_item);


                destination_of_source_connector =
                        extract_connector_destination
                                (conn_rep->menu_item);

                select_item_in_textlist(panel_schema,
                                        DESTINATIONS_FIELD,
                           destination_of_source_connector);

                rtn_memory(destination_of_source_connector);


                if (conn_rep->notes)
                        update_text_field(panel_schema,
                                          NOTES_FIELD,
                                          conn_rep->notes);
                else
                        update_text_field(panel_schema,
                                          NOTES_FIELD, "");
        }
}
else
{
        conn_rep = search_for_connector_rep_by_source
                (panel_schema, value);

        if (visible_connector_p (conn_rep))
        {
                destination_of_source_connector =
                        extract_connector_destination
                                (conn_rep->menu_item);

                if (!strcmp(destination_menu_item,
                            destination_of_source_connector))
                {
                        select_item_in_textlist
                                (panel_schema,
                                 CONNECTORS_FIELD,
                                 conn_rep->menu_item);

                        if (conn_rep->notes)
                                update_text_field
                                        (panel_schema,
                                         NOTES_FIELD,
                                         conn_rep->notes);
                        else
```

```
                                                update_text_field
                                                        (panel_schema,
                                                         NOTES_FIELD, "");
                                }

                                rtn_memory(destination_of_source_connector);
                        }
                }
        }
        else
                /* MAS: Must figure this out later */
                ;

        rtn_memory(selected_type);
}


EVENT_HANDLER CD_Selected_Destination (panel_schema, value)
art_symbol panel_schema;
char *value;
{
        char *selected_type;
        char *source_menu_item;
        char *source_type;
        char *source_of_destination_connector;
        boolean to_node, to_graph_port;
        connector_rep *conn_rep;


        if (!strcmp(value, NO_PORTS_VALUE))
                return;

        to_node = a_eq(panel_schema, NODE_TO_NODE_CONNECTOR_DETAILS_PANEL);
        to_graph_port = a_eq(panel_schema,
                             NODE_TO_GRAPH_PORT_CONNECTOR_DETAILS_PANEL);

        unselect_items_on_textlist (panel_schema, CONNECTORS_FIELD);

        selected_type = extract_type_name(value);
        if (to_node)
        {
                source_menu_item =
                        get_selected_item_from_textlist (panel_schema,
                                                         SOURCES_FIELD);

                if (source_menu_item &&
                    strcmp(source_menu_item, ""))
                {
                        source_type =
                                extract_type_name(source_menu_item);

                        conn_rep = search_for_connector_rep_by_source
                                (panel_schema, source_menu_item);

                        if (visible_connector_p (conn_rep) ||
```

```
                !compare_data_types(selected_type, source_type))
            {
                unselect_items_on_textlist(panel_schema,
                                           SOURCES_FIELD);
                source_menu_item = NULL;
            }

            rtn_memory(source_type);
        }

        if (!source_menu_item)
        {
            conn_rep = search_for_connector_rep_by_destination
                (panel_schema, value);
            if (visible_connector_p (conn_rep))
            {
                select_item_in_textlist(panel_schema,
                                        CONNECTORS_FIELD,
                                        conn_rep->menu_item);


                source_of_destination_connector =
                    extract_connector_source
                        (conn_rep->menu_item);

                select_item_in_textlist(panel_schema,
                                        SOURCES_FIELD,
                    source_of_destination_connector);

                rtn_memory(source_of_destination_connector);


                if (conn_rep->notes)
                    update_text_field(panel_schema,
                                      NOTES_FIELD,
                                      conn_rep->notes);
                else
                    update_text_field(panel_schema,
                                      NOTES_FIELD, "");
            }
        }
        else
        {
            conn_rep = search_for_connector_rep_by_destination
                (panel_schema, value);

            if (visible_connector_p (conn_rep))
            {
                source_of_destination_connector =
                    extract_connector_source
                        (conn_rep->menu_item);

                if (!strcmp(source_menu_item,
                            source_of_destination_connector))
                {
                    select_item_in_textlist
```

```
                                                (panel_schema,
                                                 CONNECTORS_FIELD,
                                                 conn_rep->menu_item);

                                        if (conn_rep->notes)
                                                update_text_field
                                                        (panel_schema,
                                                         NOTES_FIELD,
                                                         conn_rep->notes);
                                        else
                                                update_text_field
                                                        (panel_schema,
                                                         NOTES_FIELD, "");
                                }

                                rtn_memory(source_of_destination_connector);
                        }
                }
        }
        else
                /* MAS: Must figure this out later */
                ;

        rtn_memory(selected_type);
}


/*
**  EVENT_HANDLER CD_Selected_Source_Node_Details (art_symbol panel_schema);
**
**  EVENT_HANDLER CD_Selected_Destination_Node_Details
**              (art_symbol panel_schema);
**
**  These functions handle the "button has been selected" event for the
**  Source_Node_Details button and the Destination Node Details button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**  It should pop up a Node_Details_Panel for the source or destination of
**  the connector group that the Connector_Details_Panel is for.
**
**
**  COMMENTS:
**
**  * Mechanism needed to find the Source and Destination of the connector
**  group a Connector Details Panel manages.
**
********
**
**  The panel's CURRENTLY_DISPLAYED_CONNECTOR_GROUP slot contains the
**  connector group schema.  The connector group schema's SOURCE_NODE slot
```

```
**  contains the source node of the connector group and the
**  DESTINATION_NODE slot contains the destination node of the connector
**  group.
**
**  Uses populate_new_node_details_panel() to popup the new Node Details
**  Panel.
**
*/

EVENT_HANDLER CD_Selected_Source_Node_Details (panel_schema)
art_symbol panel_schema;
{
        art_symbol connector_group, source;

        connector_group =
                a_get_schema_value(panel_schema,
                                   CURRENTLY_DISPLAYED_CONNECTOR_GROUP);
        source = a_get_schema_value(connector_group, SOURCE_NODE);

        if (a_schemap(source))
                populate_new_node_details_panel(source);
}

EVENT_HANDLER CD_Selected_Destination_Node_Details (panel_schema)
art_symbol panel_schema;
{
        art_symbol connector_group, destination;

        connector_group =
                a_get_schema_value(panel_schema,
                                   CURRENTLY_DISPLAYED_CONNECTOR_GROUP);
        destination = a_get_schema_value(connector_group, DESTINATION_NODE);

        if (a_schemap(destination))
                populate_new_node_details_panel(destination);
}




/*
**  Constant To features:
*/

/*
**  EVENT_HANDLER CD_Changed_Constant_Value (art_symbol panel_schema,
**                                           char *value);
**
**  This function handles the "text field has been changed and exited"
**  event for the Constant Value field in the Constant to Node Connector
**  Details Panel.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
```

```
**              Details Panel
**
**      value is the string that was entered into the Constant Value field
**
********
**
**  If no item is selected in the Destinations textlist then do nothing.
**
**  Otherwise, value is type checked against the type of the selected item
**  in the Destinations textlist.  If the type is bad, a warning panel is
**  popped up and the previous value is restored.
**
**  If the type is ok and an item is selected in the Connections textlist,
**  then the connector item is redefined to have the new constant value
**  source and the Connections textlist is repopulated.
**
**
**  COMMENTS:
**
**      * Type checking mechanism needed
**
********
**
**  Uses extract_type_name() to get the type of the selected item
**  in the destinations textlist.  Frees the returned string when done.
**
**  Uses type_check_constant() to check the data types.
**
**  If an item is selected in the Connections textlist, then uses
**  search_for_connector_rep() to get the connector rep, from which one
**  can get to the CONNECTOR schema.  The CONNECTOR schema's SOURCE_PORT
**  slot contains the constant value which needs to be changed.
**
**
**  COMMENTS:
**
**  - The function type_check_constant() takes string data type, but
**  probably wants the DATA_TYPE schema.  How to get the schema from the
**  string is a problem.  Mentioned earlier.
**
*/

EVENT_HANDLER CD_Changed_Constant_Value (panel_schema, value)
art_symbol panel_schema;
char *value;
{}



/*
**  EVENT_HANDLER CD_Selected_Select_Constant_Value
**      (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Constant Value button on the Constant to Node Connector Details Panel.
**
```

```
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**  If no item is selected in the Destinations textlist, then popup a
**  warning panel and return.
**
**  Otherwise, popup a Select Constant Value Panel for the type of the
**  selected item in the Destinations textlist.
**
********
**
**  Uses extract_type_name() to get the type of the selected item
**  in the destinations textlist.  Frees the returned string when done.
**
**  Uses select_constant_value_modal_panel() to popup the new panel and
**  allow the user to select a value.  If the value returned is non-NULL,
**  then it is populated to the Constant Value field and the string is
**  freed.
**
**  Once the field is populated, the function CD_Changed_Constant_Value()
**  will need to be invoked.
**
**
**  COMMENTS:
**
**  - Once again, data type information is passed as a string instead of
**  as a schema.  Mentioned earlier.
**
*/

EVENT_HANDLER CD_Selected_Select_Constant_Value (panel_schema)
art_symbol panel_schema;
{}



/*
**  Graph Port features:
*/


/*
**  EVENT_HANDLER CD_Changed_Graph_Port_Name (art_symbol panel_schema,
**                                           char *value);
**
**  This function handles the "text field has been changed and exited"
**  event for the Graph Port Name field in the Graph Port to Node
**  Connector Details Panel.
**
**  Where:
**
```

```
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      value is the string that was entered into the Graph Port Name field
**
********
**
**  If no item is selected in the Destinations textlist then do nothing.
**
**  Otherwise, value is type checked against the type of the selected item
**  in the Destinations textlist.  Since graph ports can have any type,
**  the only problem can be if the graph port name is already used in a
**  connection and that connection has different type.  In this case, pop
**  up a warning panel, and restore the previous value of the Graph Port
**  Name field.
**
**  If the type is ok and an item is selected in the Connections textlist,
**  then the connector item is redefined to have the new graph port source
**  and the Connections textlist is repopulated.
**
**  (ed: Is changing graph port names desirable?  On second thought I say no.)
**
**  If no item is selected in the Destinations textlist and the Graph Port
**  is already connected to a port on the Destination node, then the
**  corresponding items in the Connections textlist and the Destinations
**  textlist are selected and the Notes field is populated with the notes
**  for that connector.
**
********
**
**  Uses type_check_graph_port() to check the graph port type.
**
**  Uses search_for_connector_rep_by_source() or
**  search_for_connector_rep_by_destination() to see if the graph port is
**  already connected.
**
**
**
*/

EVENT_HANDLER CD_Changed_Graph_Port_Name (panel_schema, value)
art_symbol panel_schema;
char *value;
{}



/*
**  EVENT_HANDLER CD_Selected_Graph_Port_Details (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Graph_Port_Details button.
**
**  Where:
**
```

```
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**  It should pop up a Grpah Port Details for the graphs input ports.
**
**
**  COMMENTS:
**
**  - Mechanism needed to find the Source and Destination of the connector
**  group a Connector Details Panel manages.
**
********
**
**  The Panel's CURRENTLY_DISPLAYED_CONNECTOR_GROUP slot contains the
**  connector group for the panel.  The connector group's ON_GRAPH slot
**  contains the graph name.
**
**  Uses populate_graph_port_details() for the new panel.  Must determine
**  if this is a Node To Graph Port or Graph Port to Node panel, for the
**  'input' argument to populate_graph_port_details().
*/

EVENT_HANDLER CD_Selected_Graph_Port_Details (panel_schema)
art_symbol panel_schema;
{}



/*****************************************************************************
*****************************************************************************
**
**   Node Details Panel
**
*/

/*
**  EVENT_HANDLER ND_Name_Changed (art_symbol panel_schema,
**                                 char *value);
**
**  This function handles the "text field has been changed and exited"
**  event for the Name field in the Node Details Panel.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Node Details Panel
**
**      value is the string that was entered into the Name field
**
********
**
**  The name associated with the node must be changed when this happens.
**
**  COMMENTS:
```

```
**
**      - Need a mechanism to change the name of a node.
**
********
**
**  The panel schema's FOR_NODE slot contains a pointer to the node.  The
**  node's NAME slot contains the name.
**
**
**  If this is for the DUMMY_ESL_EDITOR, then the DUMMY_ESL_PANEL's NODES
**  textlist must be repopulated with populate_nodes_textlist().
**
**
**  COMMENTS:
**      - A mechanism is still needed for the graphic element.
**
*/

EVENT_HANDLER ND_Name_Changed (panel_schema, value)
art_symbol panel_schema;
char *value;
{
        art_symbol node, on_graph;

        node = a_get_schema_value(panel_schema, FOR_NODE);
        on_graph = a_get_schema_value(node, ON_GRAPH);

        if (find_node_on_graph(on_graph, value))
        {
                display_warning_message(panel_schema,
"ERROR: Node by this name already exists.");
                update_text_field(panel_schema, NAME_FIELD,
                                a_string_value (a_get_schema_value(node, LABEL)));
                return;
        }
        else
                a_modify_schema_value(node, LABEL, a_art_string(value), 1L);


#ifdef DUMMY_ESL_EDITOR

        populate_nodes_textlist(DUMMY_ESL_PANEL, on_graph);

#endif DUMMY_ESL_EDITOR

}


#ifdef NOT_NEEDED_NOW

/*
**  EVENT_HANDLER ND_Selected_Input_Port (art_symbol panel_schema,
**                                        char *value);
**
**  EVENT_HANDLER ND_Selected_Output_Port (art_symbol panel_schema,
```

```
**                                              char *value);
**
**  These functions handles the "item has been selected" event for the Input
**  Ports textlist and Output Ports textlist on the Node Details Panel.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Node Details Panel
**
**      value is the string that was selected in the Input Ports or
**              Output Ports textlist
**
********
**
**  Nothing need be done on this event.
*/

EVENT_HANDLER ND_Selected_Input_Port (panel_schema, value)
art_symbol panel_schema;
char *value;
{}


EVENT_HANDLER ND_Selected_Output_Port (panel_schema, value)
art_symbol panel_schema;
char *value;
{}

#endif NOT_NEEDED_NOW


/*
**  EVENT_HANDLER ND_Selected_Input_Port_Connector_Details
**              (art_symbol panel_schema);
**
**  EVENT_HANDLER ND_Selected_Output_Port_Connector_Details
**              (art_symbol panel_schema);
**
**  These functions handle the "button has been selected" event for the
**  Input Ports Connector Details button and Output Ports Connector Details
**  button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Node Details Panel
**
*******
**
**  A Connector Details Panel is popped up for the connector group
**  corresponding to the selected item in the Input Ports or Output Ports
**  textlist.  If no item is selected or if the port is unconnected a
**  warning panel is popped up.
**
********
**
**  The panel's FOR_NODE slot contains a pointer to the node that the
```

```
**  panel is for.
**
**  The destination of the selected item is found by using
**  port_status_connected_to().  If it returns NULL, then the port is
**  unconnected and a warning panel should be popped up.
**
**  After finding the correct connector group, a Connector Details Panel
**  is popped up with populate_new_connector_details_panel().
**
**  COMMENTS:
**
**      - How to find the connector group?
**
**        Find the graph schema. Iterate over the values in its
**  HAS_CONNECTOR_GROUPS slot, until one is found that has the correct
**  source and destination.
**
*/

EVENT_HANDLER ND_Selected_Input_Port_Connector_Details (panel_schema)
art_symbol panel_schema;
{}


EVENT_HANDLER ND_Selected_Output_Port_Connector_Details (panel_schema)
art_symbol panel_schema;
{}




/*
**  EVENT_HANDLER ND_Selected_Component_Details (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Component Details button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Node Details Panel
**
********
**
**  A Component Details Panel is popped up for the node's component.
**
**  The panel's FOR_NODE slot contains a pointer to the node schema that
**  the panel is for.  If the node is a subprogram node, then the
**  component will be pointed to by the value in the USES_SUBPROGRAM slot.
**  Otherwise, the node must be an special ESL node (if, select, etc) and
**  a warning panel should be popped up.
**
**  The Component Details Panel is popped up with
**  populate_new_component_details_panel()
**
```

```
*/

EVENT_HANDLER ND_Selected_Component_Details (panel_schema)
art_symbol panel_schema;
{}



/*
**  EVENT_HANDLER ND_Selected_Notes (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Notes... button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Node Details Panel
**
********
**
**  A Node Notes Panel is popped up for the node's notes.
**
**  The panel's FOR_NODE slot conatins a pointer to the node schema that
**  the panel is for.  The Node Notes Panel is popped up with
**  populate_new_node_notes_panel()
**
*/

EVENT_HANDLER ND_Selected_Notes (panel_schema)
art_symbol panel_schema;
{}



/*
**  EVENT_HANDLER ND_Selected_Close (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Close button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Node Details Panel
**
********
**
**  Dismisses the Node Details Panel.
**
**  The panel is dismissed with dispose_of_screen()
**
*/

EVENT_HANDLER ND_Selected_Close (panel_schema)
art_symbol panel_schema;
{
        dispose_of_screen(panel_schema);
```

```
}


/*****************************************************************************
******************************************************************************
**
**   Component Details Panel
**
*/

#ifdef NOT_NEEDED_NOW

/*
**  EVENT_HANDLER C_Selected_Node_Instance (art_symbol panel_schema,
**                                         char *value);
**
**  This function handles the "item has been selected" event for the Node
**  Instances textlist on the Component Details Panel.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Component Details
**              Panel
**
**      value is the string that was selected in the Node Instances textlist
**
********
**
**  Nothing need be done on this event.
*/

EVENT_HANDLER C_Selected_Node_Instance (panel_schema, value)
art_symbol panel_schema;
char *value;
{}

#endif NOT_NEEDED_NOW



/*
**  EVENT_HANDLER C_Selected_Node_Instance_Details
**              (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Node Details button.
**
**
**  Where:
**
**      panel_schema is the schema corresponding to the Component Details
**              Panel
**
********
```

```
**
**  A Node Details Panel is popped up for the node selected in the Node
**  Instances textlist.  If no item is selected in the Node Instances
**  textlist a warning panel is popped up.
**
********
**
**  The value in the node instances textlist is the node name.  After
**  finding the corresponding node (?) the Node Details Panel is popped up
**  with populate_new_node_details_panel()
**
**  The component schema is found in the panel_schema's FOR_COMPONENT
**  slot.  To find the corresponding node schema, iterate over the values
**  in the component schema's CORRESPONDS_TO_NODES slot, then comparing
**  the value of these node schema's NAME slot with the value of the Node
**  Instances textlist.
**
*/

EVENT_HANDLER C_Selected_Node_Instance_Details (panel_schema)
art_symbol panel_schema;
{}



/*
**  EVENT_HANDLER C_Selected_Notes (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Notes... button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Component Details
**              Panel
**
********
**
**  A Component Notes Panel is popped up for the component's notes.
**
********
**
**  The panel's FOR_COMPONENT slot contains a pointer to the component
**  schema that the panel is for.  A Component Notes Panel is popped up
**  with populate_new_component_notes_panel().
**
*/

EVENT_HANDLER C_Selected_Notes (panel_schema)
art_symbol panel_schema;
{}



/*
**  EVENT_HANDLER C_Selected_Close (art_symbol panel_schema);
```

```
**
**  This function handles the "button has been selected" event for the
**  Close button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Component Details
**              Panel
**
********
**
**  Dismisses the Component Details Panel.
**
**  The panel is dismissed with dispose_of_screen()
**
*/

EVENT_HANDLER C_Selected_Close(panel_schema)
art_symbol panel_schema;
{}



/*****************************************************************************
*****************************************************************************
**
**   Graph Port Details Panel
**
*/

#ifdef NOT_NEEDED_NOW

/*
**  EVENT_HANDLER GPD_Selected_Graph_Port (art_symbol panel_schema,
**                                         char *value);
**
**  This function handles the "item has been selected" event for the Graph
**  Ports textlist on the Graph Port Details Panel.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Graph Port Details
**              Panel
**
**      value is the string that was selected in the Graph Ports textlist
**
********
**
**  Nothing need be done on this event.
*/

EVENT_HANDLER GPD_Selected_Graph_Port (panel_schema, value)
art_symbol panel_schema;
char *value;
{}
```

```
#endif NOT_NEEDED_NOW


/*
**  EVENT_HANDLER GPD_Selected_Connector_Details (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Connector Details button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Graph Port Details
**              Panel
**
********
**
**  A Node Details Panel is popped up for the node selected in the Graph
**  Ports textlist.  If no item is selected in the Graph Ports textlist or
**  if the selected port is unconnected a warning panel is popped up.
**
********
**
**  The panel's FOR_GRAPH slot contains a pointer to the graph schema that
**  the panel is for.
**
**  The destination of the selected item is found by using
**  port_status_connected_to().  If it returns NULL, then the port is
**  unconnected and a warning panel should be popped up.
**
**  After finding the correct connector group, a Connector Details Panel
**  is popped up with populate_new_connector_details_panel().
**
**  COMMENTS:
**
**      - How to find the connector group?
**        Iterate over the values in the graph schema's
**  HAS_CONNECTOR_GROUPS slot.
**
*/

EVENT_HANDLER GPD_Selected_Connector_Details (panel_schema)
art_symbol panel_schema;
{}


/*
**  EVENT_HANDLER GPD_Selected_Component_Details (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Component Details button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Graph Port Details
```

```
**              Panel
**
********
**
**  A Component Details Panel is popped up for the graph's component.
**
********
**
**  The panel's FOR_GRAPH slot contains a pointer to the graph schema that
**  the panel is for.  The Component Details Panel is popped up with
**  populate_new_component_details_panel()
**
*/

EVENT_HANDLER GPD_Selected_Component_Details (panel_schema)
art_symbol panel_schema;
{}



/*
**  EVENT_HANDLER GPD_Selected_Close (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Close button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Component Details
**              Panel
**
********
**
**  Dismisses the Graph Port Details Panel.
**
**  The panel is dismissed with dispose_of_screen()
**
*/

EVENT_HANDLER GPD_Selected_Close (panel_schema)
art_symbol panel_schema;
{}


/*****************************************************************************
*****************************************************************************
**
**   Component Notes Panel
**
*/

/*
**  EVENT_HANLDER CN_Close (art_symbol panel_schema);
```

```
**
**  This function handles the "button has been selected" event for the
**  Close button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Component Notes Panel
**
********
**
**  Dismisses the Component Notes Panel.
**
**  The panel is dismissed with dispose_of_screen()
**
*/

EVENT_HANDLER CN_Close (panel_schema)
art_symbol panel_schema;
{}



/*****************************************************************************
******************************************************************************
**
**   Node Notes Panel
**
*/


#ifdef NOT_NEEDED_NOW

EVENT_HANDLER NN_Changed_Node_Notes (art_symbol panel_schema,
                                     char **value,
                                     int numlines);

**  This function handles the "pageedit field has been changed and left"
**  event for the Node Notes field.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Node Notes Panel
**
**      value is the string that was selected in the Graph Ports textlist
**
********
**
**  Nothing need be done on this event.
*/

EVENT_HANDLER NN_Changed_Node_Notes (panel_schema, value, numlines)
art_symbol panel_schema;
char **value;
int numlines;
{}
```

```
#endif NOT_NEEDED_NOW


/*
**  EVENT_HANDLER NN_Selected_Ok (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Ok button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Node Notes Panel
**
********
**
**  Associates the current contents of the Node_Notes pageedit field with
**  the Node and dismisses the Node Notes panel.
**
**
**  COMMENTS:
**
**      - A method needs to be made to associate node notes with a
**  node.
**
********
**
**  The panel's CURRENTLY_DISPLAYED_NODE slot contains a pointer to the
**  node that the panel is for.  The node's NOTES slot contains the notes.
**
*/

EVENT_HANDLER NN_Selected_Ok (panel_schema)
art_symbol panel_schema;
{}



/*
**  EVENT_HANDLER NN_Selected_Cancel (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Cancel button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Node Notes Panel
**
********
**
**  Dismisses the Node Notes Panel.
**
**  The panel is dismissed with dispose_of_screen()
**
*/
```

```
EVENT_HANDLER NN_Selected_Cancel (panel_schema)
art_symbol panel_schema;
{}


/*****************************************************************************
******************************************************************************
**
**   Select Constant Value Panel
**
*/

#ifdef NOT_NEEDED_NOW

/*
**  EVENT_HANDLER SCV_Selected_Defined_Value (art_symbol panel_schema,
**                                            char *value);
**
**  This function handles the "item has been selected" event for the
**  Defined Values textlist on the Select Constant Value Panel.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Select
**              Constant Value Panel
**
**      value is the string that was selected in the Defined Values textlist
**
********
**
**  Nothing need be done on this event.
*/

EVENT_HANDLER SCV_Selected_Defined_Value (panel_schema, value)
art_symbol panel_schema;
char *value;
{}

#endif NOT_NEEDED_NOW



/*
**  EVENT_HANDLER SCV_Selected_Ok (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Ok button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Select
**              Constant Value Panel
**
********
**
```

```
**  If an item is selected in the Defined_Values field, then populate the
**  Constant Value field on the parental Constant To Node Connector
**  Details Panel.  Dismiss the Select Constant Value Panel.
**
**  Otherwise, popup a warning panel.
**
**  COMMENT:
**
**      * Mechanism needed to get at parental Constant To Node
**  Connector Details Panel.
**
********
**
**  The panel's HAS_PARENT_PANEL slot contains a pointer to the Constant
**  To Node panel schema that this panel emanated from.  The currently
**  selected value in the Constant Value textlist should be populated to
**  the parent panel's Constant Value Field.
**
**  Then this panel is dismissed with dispose_of_screen()
**
*/

EVENT_HANDLER SCV_Selected_Ok (panel_schema)
art_symbol panel_schema;
{}



/*
**  EVENT_HANDLER SCV_Selected_Cancel (art_symbol panel_schema);
**
**  This function handles the "button has been selected" event for the
**  Cancel button.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Select
**              Constant Value Panel
**
********
**
**  Dismisses the Select Constant Value Panel.
**
********
**
**  The panel is dismissed with dispose_of_screen()
**
*/

EVENT_HANDLER SCV_Selected_Cancel (panel_schema)
art_symbol panel_schema;
{}

/*
**
**  file:   connector_rep.c
```

```
**
**  Title:   Implementation of functions for connector rep structures, etc.
**
*/

#include "connector_rep.h"
#include "artsymbols.h"
#include "defs.h"
#include "string_utils.h"


char *strdup(/* char * */);


/* forward declarations of useful internal connector_rep functions */

void construct_initial_port_reps (/* art_symbol panel_schema,
                                     art_symbol node,
                                     boolean input */);
void enlarge_connector_reps (/* art_symbol panel_schema */);


art_symbol make_new_connector_schema (/* connector_rep *conn_rep */);


#ifdef EXTRA_FUNCTIONS_H

void find_new_connector_source_and_destination
        (/* art_symbol panel_schema,
            char *connector_menu_item,
            art_symbol *source_port
            art_symbol *destination_port);


art_symbol find_data_type (/* char *type_name */);

art_symbol find_constant_schema (/* char *type_name,
                                    char *constant_value */);

art_symbol find_port (/* art_symbol graph,
                         char *port_name,
                         boolean input */);

#endif EXTRA_FUNCTIONS_H


/*****************************************************************************/

/*
**      art_symbol find_data_type (char *type_name);
**
**  This function searches for the data type schema for the passed type
```

```
**  name.
**
**  Where
**
**      type_name is the data type to be searched for.
**
**      the return value is the data-type schema if found, NULL
**              otherwise.
**
********
**
**  Iterate over all instances of DATA_TYPE, searching for one that has
**  its NAME_OF_DATA_TYPE slot the same as the passed type name.
**
*/

art_symbol find_data_type (type_name)
char *type_name;
{}




/*
**      art_symbol find_constant_schema (char *type_name,
**                                       char *constant_value);
**
**  This function searches for a constant value schema.  If none is
**  found, then NULL is returned.
**
**  Where:
**
**      type_name is the string for the data type name
**
**      constant_value is the string of the value of the constant
**
**      the return value is the constant schema if found, NULL
**              otherwise.
**
********
**
**  First find the data_type schema with find_data_type().  Then iterate
**  over all instances of CONSTANT, searching for one that has it's
**  HAS_DATA_TYPE slot pointing to the found data type schema and it's
**  VALUE slot comparing to the passed value.
**
*/

art_symbol find_constant_schema (type_name, constant_value)
char *type_name;
char *constant_value;
{}
```

```
/*
**      art_symbol find_port (art_symbol esl_object,
**                            char *port_name,
**                            boolean input);
**
**  This function searches for a port with a particular name and
**  direction on a passed esl_object
**
**  Where:
**
**      esl_object is the graph schema or node schema who's ports are
**              to be searched
**
**      port_name is the name of the graph port to be searched for
**
**      input is TRUE if the graph's input ports are to be searched,
**              FALSE if the graph's output ports are to be searched.
**
**      the return value is the graph port schema if found, NULL
**              otherwise.
**
********
**
**  The esl_object's input ports are found in the schema's
**  HAS_INPUT_PORTS.  The output ports are found in the schema's
**  HAS_OUTPUT_PORTS.  Depending on the value of input, one of these lists
**  is searched for a port that has its LABEL slot comparing with the
**  passed port name.
**
*/

art_symbol find_port (esl_object, port_name, input)
art_symbol esl_object;
char *port_name;
boolean input;
{}




/*
**      void find_new_connector_source_and_destination
**              (art_symbol panel_schema,
**               char *connector_menu_item,
**               art_symbol *source_port,
**               art_symbol *destination_port);
**
**  This function finds the source port and destination port schemas for a
**  connection specified by its connector menu item.  If the ports don't
**  exist (in the case of graph port or constant connections), then they
**  will be created.
**
**  Where:
```

```
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      connector_menu_item is the menu item for the connection conecerned.
**
**      source_port, destination_port are pointers to art_symbols which
**              this function will set to point to the source and
**              destination objects (either port or constant schemas)
**
********
**
**  First find out the type of CD panel with determine_CD_panel_type().
**
**  Then extract the source and destination parts from the connector menu
**  item with extract_connector_source() and
**  extract_connector_destination().
**
**  Each case is handled separately:
**
********
**
**  node-to and To-Node:
**
**  Extract the port-name part from the source or destination part with
**  extract_port_name().  The panel's connector group is found in the
**  panel schema's CURRENTLY_DISPLAYED_CONNECTOR_GROUP slot.  The
**  SOURCE_NODE and DESTINATION_NODE slots of the connector group specify
**  the source and destination nodes of the connector group.
**
**  Use the function find_port() to find either the output port on the
**  source node, or the input port on the destination node.
**
********
**
**  Constant-to:
**
**  The entire source part is the constant value.  Extract the type from
**  the destination part with extract_type_name().  Use the function
**  find_constant_schema() to find the constant schema if one exists.  If
**  none exists, then create a new schema:
**
**      (defschema CONSTANT-<xx>
**              (instance-of constant)
**              (has-data-type <data-type>)
**              (value <value>))
**
********
**
**  Graph-port-to and to-graph-port:
**
**  Extract the port-name part with extract_port_name().
**
**  For graph-port-to panels, search for a graph output port and for
**  to-graph-port panels, search for a graph input port by using the
**  function find_port().  If none is found, extract the data type of the
```

```
**  "other field" (source or destination) of the passed connector menu
**  item and create a graph-port schema:
**
**      (defschema GRAPH-PORT-FOR-<graph-schema-name>-<xx>
**              (LABEL <label>)
**              (notes "")
**              (direction <dir>)
**              (port-data-type <data-type-schema>))
**
**  The caller will fill in the notes with the connector notes.
**
*/

void find_new_connector_source_and_destination
        (panel_schema, connector_menu_item,
         source_port, destination_port)
art_symbol panel_schema;
char *connector_menu_item;
art_symbol *source_port, *destination_port;
[]




/*
**      art_symbol make_new_connector_schema (connector_rep *conn_rep);
**
**  This function returns a new connector schema for the passed connector
**  rep structure.
**
**  Where
**
**      conn_rep is a pointer to the connector rep structure that the
**              new connector is for.  This should have its connector
**              membor NULL.
**
**      the return value is the new connector schema.
**
********
**
**  The big deal here is to make up a new name.  New names are of the format:
**
**      connector-<from-port>-to-<to-port>-<x>
**
**  Where <x> is generated by gentmp().
**
*/

#define CONNECTOR_NAME_FORMAT           "CONNECTOR-%s-TO-%s-"
#define CONNECTOR_NAME_FIXED_LEN        15

art_symbol make_new_connector_schema (conn_rep)
connector_rep *conn_rep;
[
        char *connector_menu_item;
```

```
        char *source, *destination;
        char *source_port_name, *destination_port_name;
        char *connector_name;
        int connector_name_len;
        art_symbol new_connector_object;

        connector_menu_item = conn_rep->menu_item;

        source = extract_connector_source (connector_menu_item);
        destination = extract_connector_destination (connector_menu_item);

        source_port_name = extract_port_name (source);
        destination_port_name = extract_port_name (destination);

        connector_name_len = CONNECTOR_NAME_FIXED_LEN +
                strlen(source_port_name) + strlen(destination_port_name);

        connector_name = (char *)
                get_memory((connector_name_len + 1) * sizeof(char));
        sprintf(connector_name, CONNECTOR_NAME_FORMAT,
                source_port_name, destination_port_name);

        new_connector_object = a_gentemp(connector_name);
        a_schemac(new_connector_object, 1L);
        a_put_schema_value(new_connector_object, INSTANCE_OF,
                           CONNECTOR, 1L);

        rtn_memory(source);
        rtn_memory(source_port_name);
        rtn_memory(destination);
        rtn_memory(destination_port_name);
        rtn_memory(connector_name);

        return new_connector_object;
}



/*
**      boolean visible_connector_p (connector_rep *conn_rep);
**
**  Tells if the passed connector rep should is real.
**
**  Where
**
**      conn_rep is a pointer to the connector_rep structure
**
********
**
**  Connectors are real if they have a schema and are associated, or
**  always if they don't have a schema.
**
*/

boolean visible_connector_p (conn_rep)
```

```
connector_rep *conn_rep;
{
        if (conn_rep)
                if (conn_rep->connector)
                {
                        if (conn_rep->associated)
                                return 1L;
                }
                else
                        return 1L;
        return 0L;
}



/*
**      connector_rep *search_for_connector_rep (art_symbol panel_schema,
**                                               char *connector_menu_item);
**
**  This function returns a pointer to the connector rep associated with
**  the passed connector_menu_item.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      connector_menu_item is the connector menu item string.  The
**              connector menu item string format is discussed in
**              make_connector_menu_item_string()
**
**      the return value is a pointer to the connector rep structure
**              that corresponds to connector_menu_item.
**
********
**
**  An pointer to an array of pointers to connector_rep structures is
**  found in the CONNECTOR_REPS slot of the passed panel schema and the
**  size of that array is stored in the CONNECTOR_REPS_SIZE slot.
**
**  For each of the connector_rep structures, the strings in the
**  connector_rep's menu_item member are compared with connector_menu_item
**  until a match is found.
**
**  If none is found, then NULL is returned.
**
*/

connector_rep *search_for_connector_rep (panel_schema, connector_menu_item)
art_symbol panel_schema;
char *connector_menu_item;
{
        connector_rep ***connector_reps;
        long connector_reps_size, i;

        connector_reps = (connector_rep ***) a_external_pointer_value
```

```
                (a_get_schema_value(panel_schema, CONNECTOR_REPS));
        connector_reps_size = a_integer_value
                (a_get_schema_value(panel_schema, CONNECTOR_REPS_SIZE));

        for (i = 0; i < connector_reps_size; i++)
                if ((*connector_reps)[i] &&
                    !strcmp((*connector_reps)[i]->menu_item,
                            connector_menu_item))
                        return (*connector_reps)[i];

        return NULL;
}


/*
**      connector_rep *search_for_connector_rep_by_source
**              (art_symbol panel_schema, char *source);
**
**      connector_rep *search_for_connector_rep_by_destination
**              (art_symbol panel_schema, char *destination);
**
**
**  These functions search through a Connector Details Panel's
**  connector_rep structures for one that has the passed source or
**  destination.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      source, destination is the value for the <source> or
**              <destination> part of a connector menu item string.
**              The connector menu item string format is discussed in
**              make_connector_menu_item_string()
**
**
**      the return value is a pointer to the connector_rep structure
**              for the connector that has the passed source or destination
**
********
**
**  A pointer to an array of pointers to connector_rep structures is found
**  in the CONNECTOR_REPS slot of the passed panel_schema and the size of
**  that array is stored in the CONNECTOR_REPS_SIZE slot.
**
**  The strings are extracted with extract_connector_source() or
**  extract_connector_destination().  The strings returned by these
**  functions is freed here.
**
*/

connector_rep *search_for_connector_rep_by_source (panel_schema, source)
art_symbol panel_schema;
char *source;
{
```

```
        connector_rep ***connector_reps;
        char *connector_rep_source;
        long connector_reps_size, i;
        boolean found;

        connector_reps = (connector_rep ***) a_external_pointer_value
                (a_get_schema_value(panel_schema, CONNECTOR_REPS));
        connector_reps_size = a_integer_value
                (a_get_schema_value(panel_schema, CONNECTOR_REPS_SIZE));

        for (i = 0; i < connector_reps_size; i++)
        {
                if ((*connector_reps)[i])
                        connector_rep_source = extract_connector_source
                                ((*connector_reps)[i]->menu_item);

                found = !strcmp(connector_rep_source, source);

                rtn_memory(connector_rep_source);
                if (found)
                        return (*connector_reps)[i];
        }

        return NULL;
}


connector_rep *search_for_connector_rep_by_destination
        (panel_schema, destination)
art_symbol panel_schema;
char *destination;
{
        connector_rep ***connector_reps;
        char *connector_rep_destination;
        long connector_reps_size, i;
        boolean found;

        connector_reps = (connector_rep ***) a_external_pointer_value
                (a_get_schema_value(panel_schema, CONNECTOR_REPS));
        connector_reps_size = a_integer_value
                (a_get_schema_value(panel_schema, CONNECTOR_REPS_SIZE));

        for (i = 0; i < connector_reps_size; i++)
        {
                if ((*connector_reps)[i])
                        connector_rep_destination =
                                extract_connector_destination
                                        ((*connector_reps)[i]->menu_item);

                found = !strcmp(connector_rep_destination, destination);

                rtn_memory(connector_rep_destination);
                if (found)
                        return (*connector_reps)[i];
        }
```

```
        return NULL;
}


/*
**      void re_associate_connector (connector_rep *conn_rep, char *notes);
**
**  This function re-associates a connector that has become unassociated
**  (ie. undefined).
**
**  Where:
**
**      conn_rep is a pointer to a connector_rep structure
**
**      notes is the new value for the connector's notes
**
********
**
**  The value of the passed connector_rep structure's associated member is
**  changed from FALSE to TRUE, and a copy of the passed notes are put
**  into the notes member.  Any previous value of the notes field is
**  freed.
**
**  It is an internal error for the associated field to be TRUE upon
**  entry.  The best thing to do in this case is to do nothing.
**
**
**  COMMENTS:
**      - The panel_schema can be used to get the notes instead of their
**  being passed to this function.
**
*/

void re_associate_connector (conn_rep, notes)
connector_rep *conn_rep;
char *notes;
{
        if (conn_rep)
        {
                conn_rep->associated = 1L;
                if (conn_rep->notes)
                        rtn_memory (conn_rep->notes);
                conn_rep->notes = strdup(notes);
        }
}


/*
**      void disassociate_connector_rep (art_symbol panel_schema,
**                                       connector_rep *conn_rep);
**
**  This function disassociates a connector (ie: undefines it.).
**
**  Where:
```

```
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      conn_rep is a pointer to a connector_rep structure
**
********
**
**  If the connector member of conn_rep* is non-NULL, then the connector
**  has previously existed.  The assoicated member is set to FALSE.
**
**  If the connector member of conn_rep* is NULL, then the connector
**  object has never been made.  The panel schema's CONNECTOR_REPS slot
**  contains a pointer to an array of pointers to connector_rep structures
**  and the CONNECTOR_REPS_SIZE slot contains the size of that array.
**  This array is searched for a pointer that corresponds to conn_rep.  If
**  the notes member of conn_rep* is non-NULL, then the string it points
**  to is freed.  The memory used by the conn_rep* is also freed.  The
**  pointer to conn_rep in the CONNECTOR_REPS array is made NULL.
**
*/

void disassociate_connector_rep (panel_schema, conn_rep)
art_symbol panel_schema;
connector_rep *conn_rep;
{
        connector_rep ***connector_reps;
        long connector_reps_size, i;

        if (!conn_rep)
                return;

        if (conn_rep->connector)
                conn_rep->associated = 0L;
        else
        {
                connector_reps = (connector_rep ***) a_external_pointer_value
                        (a_get_schema_value(panel_schema, CONNECTOR_REPS));
                connector_reps_size = a_integer_value
                        (a_get_schema_value(panel_schema,
                                            CONNECTOR_REPS_SIZE));

                for (i = 0; i < connector_reps_size; i++)
                        if ((*connector_reps)[i] &&
                            (*connector_reps)[i] == conn_rep)
                        {
                                (*connector_reps)[i] = NULL;
                                if (conn_rep->notes)
                                        rtn_memory(conn_rep->notes);
                                rtn_memory(conn_rep);

                                break;
                        }
        }
}
```

```
/*
**      void construct_initial_reps (art_symbol panel_schema);
**
**  This function constructs the connector_rep structures and port_rep
**  structures for the passed Connector Details panel schema.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**  Unless the panel is a Constant To panel, the input port reps are
**  constructed using construct_initial_port_reps().  The output port reps
**  are always constructed using the same function.
**
**  The connector_reps are constructed using
**  construct_initial_connector_reps()
**
*/

void construct_initial_reps (panel_schema)
art_symbol panel_schema;
{}



/*
**      void construct_initial_port_reps (art_symbol panel_schema,
**                                        art_symbol node,
**                                        boolean input)
**
**  This function creates the support port_rep structures for Connector
**  Details Panels.  Only one set (input or output) is made with one call
**  to this funciton.  If the panel is a Node To Node Connector Details
**  panel this function will have to be called twice.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel.
**
**      node is the node schema that the ports are attached to.
**
**      input is TRUE if the node's input ports are to be used, FALSE
**              if the node's output ports are to be used.
**
**
**  Once this is done, the panel-schema may be fed into
**  search_for_port_rep()
**
********
```

```
**
**  The node's input ports are found in the node schema's HAS_INPUT_PORTS
**  slot; the output ports are found in the node schema's HAS_OUTPUT_PORTS
**  slot.  The a pointer to an array of pointers to port_rep structures
**  for input ports is put in the panel-schema's INPUT_PORT_REPS slot, and
**  for output ports it is put in the panel-schema's OUTPUT_PORT_REPS
**  slot.
**
**  Memory is allocated for the an array of pointers to port reps that is
**  one larger than the number of ports, and also for a port_rep structure
**  for each port.  This memory should be allocated all at once so that it
**  can be freed all at once.
**
**  The array is initialized with a pointer to each successive port_rep
**  structure, terminating with a NULL pointer.
**
**  The port_rep structures are given values by iterating over the ports
**  and:
**
**      filling the port member with a pointer to the port schema
**
**      filling the menu_item member with a port-type menu item
**  created by calling create_port_type_menu_item() with the value of the
**  port's NAME slot and either the value of the port's PORT_DATA_TYPE (if
**  it is a string) or use that value to find the data-type schema and use
**  it's NAME_OF_DATA_TYPE slot.
**
*/

void construct_initial_port_reps (panel_schema, node, input)
art_symbol panel_schema;
art_symbol node;
boolean input;
{}



/*
**      void construct_initial_connector_reps (art_symbol panel_schema,
**                                             art_symbol connector_group);
**
**  This function creates the support connector_rep structures for
**  Connector Details Panels.  The necessary port_rep structures should be
**  made first with construct_initial_port_reps()
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel.
**
**      connector_group is the connector-group schema that the
**              Connector Details Panel displays
**
********
**
**  Any previous values in the CONNECTOR_REPS and CONNECTOR_REPS_SIZE
```

```
**  slots are reused.  If none exists or if the array needs to be resized,
**  the function enlarge_connector_reps() is used.
**
**  The connector group that the panel is for is found in the
**  CURRENTLY_DISPLAYED_CONNECTOR_GROUP slot.  The HAS_CONNECTORS slot in
**  the connector-group schema contains the list of connector objects.
**  This list is iterated over and the connector_rep structures are filled
**  in by add_new_connector_rep().  The connector menu items are
**  constructed with make_connector_menu_item().
**
**  The connector schema's SOURCE_PORT slot contains a pointer to the
**  source port schema of the connector and the DESTINATION_PORT slot
**  contains a pointer to the destination port schema.
**
**  If the panel is a Node To panel or Graph Port To panel then the source
**  argument to make_connector_menu_item() is found by calling
**  search_for_port_rep_by_port() with the port schema and using it's
**  menu_item member.  If the panel is a To Node or To Graph Port panel
**  then the destination argument is found similarily.
**
**  On Constant To panels, one must use the constant value for the source
**  argument to make_connector_menu_item().  The constant schema is
**  pointed to by the connector schema's SOURCE_PORT slot, and the value
**  is stored in its VALUE slot.
**
*/

static art_object connector_details_panel;
static art_object this_connector_group;
static boolean constant_source;

boolean construct_initial_connector_rep (connector)
art_object connector;
{
        art_symbol source, destination;
        char *source_substring, *destination_substring;
        char *connector_menu_item;
        char *notes;

        source = a_get_schema_value(connector, SOURCE_PORT);
        destination = a_get_schema_value(connector, DESTINATION_PORT);

        if (a_slot_null(connector, NOTES))
                notes = NULL;
        else
                notes = a_string_value(a_get_schema_value(connector, NOTES));

        if (constant_source)
        {
                if (a_stringp(source))
                        source_substring = a_string_value(source);
                else
                        source_substring = a_string_value
                                (a_get_schema_value(source, VALUE));
        }
        else
```

```
                source_substring = make_port_type_menu_item (source);

        destination_substring = make_port_type_menu_item (destination);

        connector_menu_item = make_connector_menu_item(source_substring,
                                                       destination_substring);

        add_new_connector_rep (connector_details_panel,
                               connector_menu_item,
                               connector, notes);

        if (!constant_source)
                rtn_memory (source_substring);
        rtn_memory (destination_substring);

        return 1L;
}



void construct_initial_connector_reps (panel_schema, connector_group)
art_symbol panel_schema;
art_symbol connector_group;
{
        connector_details_panel = panel_schema;
        this_connector_group = connector_group;

        constant_source =
                a_eq(a_get_schema_value(connector_group, SOURCE_NODE),
                     CONSTANT_VALUE);

        a_for_schema_values(connector_group, HAS_CONNECTORS,
                            construct_initial_connector_rep, NULL);
}



/*
**      connector_rep *add_new_connector_rep (art_symbol panel_schema,
**                                            char *connector_menu_item,
**                                            art_symbol connector,
**                                            char *notes)
**
**  This function makes a new connector_rep structure for a new
**  connection.  It returns a pointer to the new connector_rep structure.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      connector_menu_item is a connector menu item string
**
**      connector is the corresponding connector schema
```

```
**
**      notes is the notes for the connector
**
**      the return value is a pointer to the connector rep structure
**              made.
**
********
**
**  If connector is NULL, then the connector_rep is for a newly defined
**  connector.  This is the case when this function is called from
**  CD_Select_Connect().
**
**  If connector is non-NULL, then the connector_rep is for an already
**  existing connector.  This is the case when it is called by
**  construct_initial_reps().
**
**
**  The array of connector_rep pointers is stored in the panel_schema's
**  CONNECTOR_REPS slot and the current size of that array is stored in
**  the CONNECTOR_REPS_SIZE slot.  This array is scanned for a NULL
**  pointer.  If no NULL pointer is found, then the array is enlarged as
**  follows:
**
**      1. Allocate memory a larger array of pointers.  Change the
**  CONNECTOR_REPS_SIZE slot.  The array is size increased by some
**  constant value (ed: specify constant value later).
**
**      2. Copy the old array of pointers.
**
**      3. Free up the old array of pointers.
**
**
**  A new connector_rep structure is allocated and fields are filled in as
**  follows:
**
**      1. The menu_item and notes members are set to the passed
**  connector_menu_item and a copy of notes.
**
**      2. The connector member is given the passed connector value.
**  The associated member is set to TRUE.
**
**  If this function fails (only possible if memory cannot be allocated),
**  it will return NULL.
**
**
**  COMMENTS:
**
**      - Failure is so unexpected, that maybe it should raise a warning.
**
**      - May want to break up enlarge array functionality into another
**      function.
**
********
**
**  Regarding enlarging the array of pointers to connector_rep structures:
**
```

```
**      If there is no value on the panel schema's CONNECTOR_REPS
**  slot, then a new array of pointers to connector_reps is a allocated
**  with size MINIMUM_CONNECTOR_REPS_SIZE.  The array is initialized with
**  NULL pointers.  The panel schema's CONNECTOR_REPS slot and
**  CONNECTOR_REPS_SIZE slots are filled in with a pointer to this array
**  and the size of the array respectively.
**
**      If the panel already has connector_reps, then a new array is
**  allocated with size equal to the previous value in the
**  CONNECTOR_REPS_SIZE slot plus the constant
**  ENLARGEMENT_CONNECTOR_REPS_SIZE.  The old array of pointers is copied
**  into the new array and then freed.  The CONNECTOR_REPS and
**  CONNECTOR_REPS_SIZE slots are also updated.
**
*/

connector_rep *add_new_connector_rep (panel_schema,
                                      connector_menu_item, connector, notes)
art_symbol panel_schema;
char *connector_menu_item;
art_symbol connector;
char *notes;
{
        connector_rep *new_connector_rep;
        connector_rep ***old_connector_reps, ***new_connector_reps;
        long connector_reps_size, new_connector_reps_size;
        long old_connector_reps_size;
        int i;

        new_connector_rep = (connector_rep *)
                (get_memory(1 * sizeof(connector_rep)));

        new_connector_rep->menu_item = connector_menu_item;
        new_connector_rep->connector = connector;
        new_connector_rep->associated = (connector ? 1L : 0L);

        if (notes)
                new_connector_rep->notes = strdup(notes);
        else
                new_connector_rep->notes = NULL;

        if (a_slot_null(panel_schema, CONNECTOR_REPS))
        {
                new_connector_reps = (connector_rep ***)
                        get_memory(1 * sizeof(connector_rep **));

                *new_connector_reps = (connector_rep **)
                        get_memory(MINIMUM_CONNECTOR_REPS_SIZE
                                * sizeof(connector_rep *));

                a_modify_schema_value(panel_schema, CONNECTOR_REPS,
                                a_art_external_pointer(new_connector_reps),
                                        1L);
                a_modify_schema_value(panel_schema, CONNECTOR_REPS_SIZE,
                                    a_art_integer(MINIMUM_CONNECTOR_REPS_SIZE),
                                        1L);
```

```
                (*new_connector_reps)[0] = new_connector_rep;

                for (i = 1; i < MINIMUM_CONNECTOR_REPS_SIZE; i++)
                        (*new_connector_reps)[i] = NULL;

                return;
        }

        old_connector_reps = (connector_rep ***) a_external_pointer_value
                (a_get_schema_value(panel_schema, CONNECTOR_REPS));
        connector_reps_size = a_integer_value
                (a_get_schema_value(panel_schema, CONNECTOR_REPS_SIZE));

        for (i = 0; i < connector_reps_size; i++)
                if (!(*old_connector_reps)[i])
                {
                        (*old_connector_reps)[i] = new_connector_rep;
                        return;
                }


        /* otherwise must enlarge array... */

        new_connector_reps = (connector_rep ***)
                get_memory(1 * sizeof(connector_rep **));
        new_connector_reps_size =
                connector_reps_size + ENLARGEMENT_CONNECTOR_REPS_SIZE;

        a_modify_schema_value(panel_schema, CONNECTOR_REPS,
                                a_external_pointer_value(new_connector_reps));
        a_modify_schema_value(panel_schema, CONNECTOR_REPS_SIZE,
                                a_art_integer(MINIMUM_CONNECTOR_REPS_SIZE));

        *new_connector_reps = (connector_rep **)
                get_memory(new_connector_reps_size
                           * sizeof(connector_rep *));

        for (i = 0; i < connector_reps_size; i++)
                (*new_connector_reps)[i] = (*old_connector_reps)[i];

        for (i = connector_reps_size;
             i < new_connector_reps_size;
             i++)
                (*new_connector_reps)[i] = NULL;

        (*new_connector_reps)[connector_reps_size] = new_connector_rep;

        rtn_memory(*old_connector_reps);
        rtn_memory(old_connector_reps);
}


/*
```

```
**      void free_up_reps (art_symbol panel_schema);
**
**  This function frees up memory used by the panel_schema's connector_rep
**  and port_rep structures, prior to dismissing the panel.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**
**  If the panel is not a Constant To panel, a pointer to an array of
**  pointers to port_reps are found in the panel schema's INPUT_PORT_REPS
**  slot.  This memory is freed here.  All panlels have an
**  OUTPUT_PORT_REPS slot which points to memory which must be similarily
**  freed.
**
**  This function iterates over the array of pointers to connector_rep
**  structures found in the panel_schema's CONNECTOR_REPS slots.  The size
**  of the array is stored in the CONNECTOR_REPS_SIZE slot.
**
**  All connector_rep structs pointed to by the CONNECTOR_REPS array have
**  their menu_item and notes member strings freed.  Then the memory used
**  by the connector_rep structure is then freed and the pointers in the
**  array are set to NULL.
**
*/

void free_up_reps (panel_schema)
art_symbol panel_schema;
{
        connector_rep ***connector_reps, *this_connector_rep;
        long connector_reps_size;
        long i;

        connector_reps = (connector_rep ***) a_external_pointer_value
                (a_get_schema_value(panel_schema, CONNECTOR_REPS));
        connector_reps_size = a_integer_value
                (a_get_schema_value(panel_schema, CONNECTOR_REPS_SIZE));

        for (i = 0; i < connector_reps_size; i++)
        {
                this_connector_rep = (*connector_reps)[i];

                if (this_connector_rep)
                {
                        rtn_memory(this_connector_rep->menu_item);
                        if (this_connector_rep->notes)
                                rtn_memory(this_connector_rep->notes);
                        rtn_memory(this_connector_rep);

                        (*connector_reps)[i] = NULL;
                }
        }
```

```
}


/*
**      void create_and_destroy_changed_connectors (art_symbol panel_schema);
**
**  This function creates a connector schema for each newly defined
**  connector, modifies connectors with changed definitions, and destroys
**  newly undefined connectors.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**  It iterates over the array of pointers to connector_reps found in the
**  CONECTOR_REPS slot of the passed panel_schema.  The size of that array
**  is found in the CONNECTOR_REPS_SIZE slot.
**
**  For each connector rep structure pointed to:
**
**      - If the value of connector member is non-NULL and associated
**  is FALSE, then the corresponding connector schema must be destroyed.
**  This requires that:
**
**              1. The connector group schema no longer reference it.
**
**              2. The memory used by the menu_item and notes fields
**  is freed.  The memory used by the connector_rep is also freed.  The
**  pointer to the connector_rep structure is set to NULL.
**
**              3. The connector schema be deleted
**
**
**      - If the value of the connector member is non-NULL and
**  associated is TRUE, then the corresponding connector schema may need
**  to have their NOTES slots changed.  The notes member contains current
**  notes.
**
**
**      - If the value of connector member is NULL and associated is
**  TRUE, then a new connector object must be created.  This requires
**  that:
**
**              1. A new connector schema is made with NOTES and
**  BELONGS_TO_CONNECTOR_GROUP slots filled in.
**
**              2. The connector group must reference this new
**  connector object in its HAS_CONNECTORS slot.
**
**              3. The connector field in the connector_rep must point
**  the new connector object.
**
```

```
MAS: Put in something for the ports...

*/

void create_and_destroy_changed_connectors (panel_schema)
art_symbol panel_schema;
{
        connector_rep ***connector_reps, *this_connector_rep;
        long connector_reps_size, i;
        art_symbol this_connector, connector_group;

        connector_reps = (connector_rep ***) a_external_pointer_value
                (a_get_schema_value(panel_schema, CONNECTOR_REPS));
        connector_reps_size = a_integer_value
                (a_get_schema_value(panel_schema, CONNECTOR_REPS_SIZE));

        for (i = 0; i < connector_reps_size; i++)
        {
                this_connector_rep = (*connector_reps)[i];
                if (!this_connector_rep)
                        continue;
                this_connector = this_connector_rep->connector;

                if (this_connector && !this_connector_rep->associated)
                {
                        /* destroy an old connector object... */

                        /* retract the BELONGS_TO_CONNECTOR_GROUP */
                        connector_group = a_get_schema_value
                                (this_connector,
                                 BELONGS_TO_CONNECTOR_GROUP);

                        a_retract_schema_value(connector_group,
                                               HAS_CONNECTORS,
                                               this_connector, 1L);

                        /* Return memory */
                        rtn_memory(this_connector_rep->menu_item);
                        if (this_connector_rep->notes)
                                rtn_memory(this_connector_rep->notes);
                        rtn_memory(this_connector_rep);

                        (*connector_reps)[i] = NULL;

                        /* delete the schema */
                        a_schemad(this_connector);
                }
                else if (this_connector && this_connector_rep->associated)
                {
                        /* update existing connector object... */

                        if (this_connector_rep->notes)
                                a_modify_schema_value
                                        (this_connector,
```

```
                                        NOTES,
                                    a_art_string(this_connector_rep->notes),
                                        1L);
                        else
                                a_modify_schema_value
                                        (this_connector,
                                        NOTES, a_art_string(""), 1L);
                }
                else
                {
                        /* create new connector object... */
                        this_connector =
                                make_new_connector_schema(this_connector_rep);

                        connector_group = a_get_schema_value
                                (panel_schema,
                                 CURRENTLY_DISPLAYED_CONNECTOR_GROUP);

                        a_modify_schema_value(this_connector,
                                              BELONGS_TO_CONNECTOR_GROUP,
                                              connector_group, 1L);
                        a_put_schema_value(connector_group,
                                           HAS_CONNECTORS,
                                           this_connector, 1L);


                        if (this_connector_rep->notes)
                                a_modify_schema_value
                                        (this_connector,
                                         NOTES,
                                    a_art_string(this_connector_rep->notes),
                                         1L);
                        else
                                a_modify_schema_value
                                        (this_connector,
                                         NOTES, "", 1L);

                        this_connector_rep->connector = this_connector;
                }
        }
}



/*
**      port_rep *search_for_port_rep_by_menu_item (art_symbol panel_schema,
**                                                  char *port_type_menu_item,
**                                                  boolean input);
**
**      port_rep *search_for_port_rep_by_port (art_symbol panel_schema,
**                                            art_symbol port,
**                                            boolean input);
**
**  These functions searche for the port_rep structure that corresponds to
**  a port-type menu item or port schema, given the port direction.
```

```
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      port_type_menu_item is the port-type menu item to be searched
**              for.  port-type menu item format is discussed in
**              construct_port_type_menu_item()
**
**      port is the port schema
**
**      input is TRUE if the port is an input port, FALSE otherwise
**
********
**
**  If input is TRUE, the panel schema's INPUT_PORT_REPS slot is used for
**  the search.  Otherwise, the panel schema's OUTPUT_PORT_REPS slot is
**  used.
**
**  The panel schema's [INPUT|OUTPUT]_PORT_REPS slot contains a pointer to
**  a NULL terminated array of pointers to port_rep structures.
**
**  search_for_port_rep_by_menu_item() searches these port_rep structures
**  for one that has its menu_item member that strcmp() with the passed
**  port-type menu item.
**
**  search_for_port_rep_by_port compares the port_rep's port member with
**  the passed port (ed: with a_eq or = ?).
**
**
**  COMMENTS:
**
**  - It is an internal error to for the panel schema not to have the
**  proper slot, or for there to be no value in this slot.  Since it is
**  probably caused by a logic flaw we will want fixed, a warning message
**  should be generated.
**
*/

port_rep *search_for_port_rep_by_menu_item (panel_schema,
                                            port_type_menu_item, input)
art_symbol panel_schema;
char *port_type_menu_item;
boolean input;
{}

port_rep *search_for_port_rep_by_port (panel_schema, port, input)
art_symbol panel_schema;
art_symbol port;
boolean input;
{}


/*
```

```
**  file: extra_functions.c
**
**  Function bodies for extra support stuff for ESL.  All functions are
**  grouped here for now.  That'll be changed soon.
**
*/


#include "extra_functions.h"



/* string_utils.c */

/*
**      char *make_connector_menu_item (char *source, char *destination);
**
**  This function makes a connector menu item for the passed source and
**  destination.
**
**  Where:
**
**      source, destination are the strings to be the <source> and
**              <destination> part of the connector menu item
**
**      the return value is the generated connector menu item string
**
********
**
**  This function allocates memory for the returned string which must be
**  freed later.
**
**  The generated connector menu item will look like:
**
**      "<source> to <destination>"
**
**  The <source> and <destination> fields will have some minimum constant
**  width (ed: specificy constant name later).  The " to " may need more
**  padding spaces to align the view properly.
**
**
**  COMMENTS:
**      - Possibly the source and destination arguments could be art_symbols.
**
*/

char *make_connector_menu_item (source, destination)
char *source, *destination;
{}



/*
**      char *extract_connector_source (char *connector_menu_item);
**      char *extract_connector_destination (char *connector_menu_item);
**
```

```
**  These functions take a connector menu item and extracts either the
**  source or the destination part.
**
**  Where:
**
**      connector_menu_item is the connector menu item string
**
**      the return value is the <source> or <destination> extraction
**              from the connector menu item string.
**
********
**
**
**  The connector menu item string format is discussed in
**  make_connector_menu_item().  These functions return either the
**  <source> or <destination> part with no trailing spaces.
**
**  The source string is suitable to be populated to the Sources textlist
**  on Node To panels, the Destinations textlist on To Node panels and
**  Constant Value field on Constant To panels.  It must have the type
**  information stripped off with extract_port_name() for the the Graph
**  Port Name field on Graph Port To and To Graph Port panels.
**
**  These functions allocate memory for the returned string which must be
**  freed later.
**
*/

char *extract_connector_source (connector_menu_item)
char *connector_menu_item;
{}


char *extract_connector_destination (connector_menu_item)
char *connector_menu_item;
{}


/*
**      char *make_port_type_menu_item (char *port, char *type);
**
**  This function returns a pointer to a port_type menu item string for
**  the passed port and data type.
**
**  Where:
**
**      port, type are the strings to be the <port> and <type> part of
**              the port-type menu item
**
**      the return value is the generated port-type menu item string
**
********
**
**  The generated port-type menu item will look like:
**
**      "<port> : <type>"
```

```
**
**
**  The <port> and <type> fields will have some minimum constant width
**  (ed: specify constant names later).  The ":" will always have at least
**  one space on each side.
**
**  This function allocates memory for the returned string which must be
**  freed later.
**
*/

char *make_port_type_menu_item (char *port, char *type)
char *port, *type;
{}


/*
**      char *extract_port_name (char *port_type_menu_item);
**      char *extract_type_name (char *port_type_menu_item);
**
**  These functions take a port_type menu item and extract either the port
**  name or data type part.
**
**  Where:
**
**      port_type_menu_item is the port-type menu item string
**
**      the return value is the <port> or <type> extraction from the
**              port-type menu item string
**
********
**
**  The port-type menu item string format is discussed in
**  make_port_type_menu_item().  These functions return either the <port>
**  or <type> part with no trailing spaces.
**
**  They allocate memory for the returned string which must be freed
**  later.
**
*/

char *extract_port_name (port_type_menu_item)
char *port_type_menu_item;
{}


char *extract_type_name (port_type_menu_item)
char *port_type_menu_item;
{}



/*
**      char *make_component_port_status_menu_item (art_symbol port);
**      char *make_node_port_status_menu_item (art_symbol port);
```

```
**
**
**  These functions generate a port status menu item string for a
**  particular context.
**
**  Where:
**
**      port is the port schema that the menu item is to be generated for.
**
**      the return value is the generated port-status menu item string
**
********
**
**  make_component_port_status_menu_item() is used to generate menu items
**  for the Input Ports and Output Ports textlists on the Component
**  Details panel.
**
**  make_node_port_status_menu_item() is used to generate menu items for
**  the Input Ports and Output Ports textlists on the Node Details panel,
**  and the Graph Ports textlist on the Graph Ports Details panel.
**
**
**  The generated component-port-status menu item looks like:
**
**      "<name> : <type> -- <notes>"
**
**  The generated node-port-status and menu item looks like:
**
**      "<name> : <type> <status>"
**
**  Where <status> is:
**
**      "<direction> <connected-node>, <connected-port>"
**
**  if the port is connected and:
**
**      "unconnected"
**
**  otherwise.
**
**
**  Where:
**
**      <name> is the port name, found in the port schema's NAME slot.
**             There is some minimum width for this field (ed:
**             specify constant value later.)
**
**      <type> is the port data type, found either in the port
**             schema's PORT-DATA-TYPE slot (if a string value), or
**             in the data-type schema's NAME_OF_DATA_TYPE slot (if a
**             symbol value).
**
**      <notes> is the port notes found in the port schema's NOTES slot.
**
**      <direction> is either " from " or " to " depending on the
**             value in the port schema's DIRECTION slot.
```

```
**
**      <connected-node> and <connected-port> are the values of the
**              connected node schema's NAME slot and the connected
**              port schema's NAME slot.
**
**  The <name>, <type> fields have a minimum width (ed: specify constant
**  value later).  The ":" and "--" are always have at least one space on
**  either side.  The "," never has a leading space and always has a
**  trailing space.
**
**
**  The connected node and connected port are found as follows:
**
**  The port's node is found in the port schema's ON_NODE slot.  The graph
**  that the node is on is in found in the node schema's ON_GRAPH slot.
**  The graph schema's HAS_CONNECTOR_GROUPS slot lists the the connector
**  groups on the graph.
**
**  If the value of port schema's DIRECTION slot is INPUT, the list of
**  connector group schemas is searched for one that has this node in its
**  SOURCE_NODE slot; the value in the connector group schema's
**  DESTINATION_NODE slot is the connecting node.  The connector group
**  schema's HAS_CONNECRTORS slot lists the connectors in the connector
**  group.  This list is searched for one that has the passed port in its
**  SOURCE_PORT slot.  The value in the found connector schema's
**  DESTINATION_PORT slot is the destination port.  If there is no
**  connector group schema that has this node as its SOURCE_NODE, or there
**  is no connector schema that has this port as its SOURCE_PORT, then the
**  port is unconnected.
**
**  A similar algorithm is used if the value of the port schema's
**  DIRECTION slot is OUTPUT.
**
**  if the port schema's direction slot is OUTPUT, the
**  list of connector group schemas is searched fro one that has this node
**  in its DESTINATION_NODE slot.
**
**
**  These functions allocate memory which must be freed later.
**
*/

char *make_component_port_status_menu_item (art_symbol node_port)
art_symbol node_port;
{}


char *make_node_port_status_menu_item (art_symbol node_port)
art_symbol node_port;
{}



/*
**      art_symbol port_status_connected_to (char *port_status_menu_item);
```

```
**
**  This function takes in a port-status menu item and returns the port
**  object that is connected to the port.  If the port is unconnected,
**  then this function returns NULL
**
**  Where:
**
**      port_status_menu_item is the port-status menu item string
**
**      the return value is the port schema that the port-status menu
**              item indicates a connection to, or NULL
**
********
**
**  MAJOR COMMENT:
**
**  - Q: How does this work?
**    A: It probably doesn't, and will need redesign.
**
*/

art_symbol port_status_connected_to (port_status_menu_item)
char *port_status_menu_item;
{}



/*
**      char *esl_object_type_string (art_symbol esl_object);
**
**      char *esl_implementation_type_string (art_symbol esl_object);
**
**  These functions return a string that can be populated to a Type or
**  Implementation Type static text field.
**
**  Where:
**
**      esl_object is an esl object schema that is an instance of
**              SUBPROGRAM or NODE.
**
**      the return value is the string that contains the description
**              of the type of esl_object
**
********
**
**  The string is a statically constant allocated constant and may not be
**  freed.
**
**  For example, the string returned for PRIMITIVE_SUBPROGRAM_NODE is
**  "Application Procedure", for IF_NODE is "If Node", etc.
**
*/

char *esl_object_type_string (esl_object)
art_symbol esl_object;
{}
```

```
char *esl_implementation_type_string (art_symbol esl_object)
art_symbol esl_object;
{}


/* connector_rep.c */

/* forward declarations of useful internal connector_rep functions */

void construct_initial_connector_reps (/* art_symbol panel_schema */);

void construct_initial_port_reps (/* art_symbol panel_schema,
                                     art_symbol node,
                                     boolean input */);



/*
**      connector_rep *search_for_connector_rep (art_symbol panel_schema,
**                                               char *connector_menu_item);
**
**  This function returns a pointer to the connector rep associated with
**  the passed connector_menu_item.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      connector_menu_item is the connector menu item string.  The
**              connector menu item string format is discussed in
**              make_connector_menu_item_string()
**
**      the return value is a pointer to the connector rep structure
**              that corresponds to connector_menu_item.
**
********
**
**  An pointer to an array of pointers to connector_rep structures is
**  found in the CONNECTOR_REPS slot of the passed panel schema and the
**  size of that array is stored in the CONNECTOR_REPS_SIZE slot.
**
**  For each of the connector_rep structures, the strings in the
**  connector_rep's menu_item member are compared with connector_menu_item
**  until a match is found.
**
**  If none is found, then NULL is returned.
**
*/
```

```
connector_rep *search_for_connector_rep (panel_schema, connector_menu_item)
art_symbol panel_schema;
char *connector_menu_item;
{}


/*
**      connector_rep *search_for_connector_rep_by_source
**              (art_symbol panel_schema, char *source);
**
**      connector_rep *search_for_connector_rep_by_destination
**              (art_symbol panel_schema, char *destination);
**
**
**  These functions search through a Connector Details Panel's
**  connector_rep structures for one that has the passed source or
**  destination.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      source, destination is the value for the <source> or
**              <destination> part of a connector menu item string.
**              The connector menu item string format is discussed in
**              make_connector_menu_item_string()
**
**
**      the return value is a pointer to the connector_rep structure
**              for the connector that has the passed source or destination
**
********
**
**  A pointer to an array of pointers to connector_rep structures is found
**  in the CONNECTOR_REPS slot of the passed panel_schema and the size of
**  that array is stored in the CONNECTOR_REPS_SIZE slot.
**
**  The strings are extracted with extract_connector_source() or
**  extract_connector_destination().  The strings returned by these
**  functions is freed here.
**
*/

connector_rep *search_for_connector_rep_by_source (panel_schema, source)
art_symbol panel_schema;
char *source;
{}


connector_rep *search_for_connector_rep_by_destination
        (panel_schema, destination)
art_symbol panel_schema;
char *destination;
{}
```

```
/*
**      void re_associate_connector (connector_rep *conn_rep, char *notes);
**
**  This function re-associates a connector that has become unassociated
**  (ie. undefined).
**
**  Where:
**
**      conn_rep is a pointer to a connector_rep structure
**
**      notes is the new value for the connector's notes
**
********
**
**  The value of the passed connector_rep structure's associated member is
**  changed from FALSE to TRUE, and a copy of the passed notes are put
**  into the notes member.  Any previous value of the notes field is
**  freed.
**
**  It is an internal error for the associated field to be TRUE upon
**  entry.  The best thing to do in this case is to do nothing.
**
**
**  COMMENTS:
**      - The panel_schema can be used to get the notes instead of their
**  being passed to this function.
**
*/

void re_associate_connector (conn_rep, notes)
connector_rep *conn_rep;
char *notes;
{}



/*
**      void disassociate_connector_rep (art_symbol panel_schema,
**                                       connector_rep *conn_rep);
**
**  This function disassociates a connector (ie: undefines it.).
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      conn_rep is a pointer to a connector_rep structure
**
********
**
**  If the connector member of conn_rep* is non-NULL, then the connector
**  has previously existed.  The assoicated member is set to FALSE and if
**  the notes member is non-NULL, the string it points to is freed and the
```

```
**  notes member is made NULL.
**
**  If the connector_member of conn_rep* is NULL, then the connector
**  object has never been made.  The panel schema's CONNECTOR_REPS slot
**  contains a pointer to an array of pointers to connector_rep structures
**  and the CONNECTOR_REPS_SIZE slot contains the size of that array.
**  This array is searched for a pointer that corresponds to conn_rep.  If
**  the notes member of conn_rep* is non-NULL, then the string it points
**  to is freed.  The memory used by the conn_rep* is also freed.  The
**  pointer to conn_rep in the CONNECTOR_REPS array is made NULL.
**
*/

void disassociate_connector_rep (panel_schema, conn_rep)
art_symbol panel_schema;
connector_rep *conn_rep;
{}



/*
**      void construct_initial_reps (art_symbol panel_schema);
**
**  This function constructs the connector_rep structures and port_rep
**  structures for the passed Connector Details panel schema.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**  Unless the panel is a Constant To panel, the input port reps are
**  constructed using construct_initial_port_reps().  The output port reps
**  are always constructed using the same function.
**
**  The connector_reps are constructed using
**  construct_initial_connector_reps()
**
*/

void construct_initial_reps (panel_schema)
art_symbol panel_schema;
{}



/*
**      void construct_initial_port_reps (art_symbol panel_schema,
**                                        art_symbol node,
**                                        boolean input)
**
**  This function creates the support port_rep structures for Connector
**  Details Panels.  Only one set (input or output) is made with one call
```

```
**  to this funciton.  If the panel is a Node To Node Connector Details
**  panel this function will have to be called twice.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel.
**
**      node is the node schema that the ports are attached to.
**
**      input is TRUE if the node's input ports are to be used, FALSE
**              if the node's output ports are to be used.
**
**
**  Once this is done, the panel-schema may be fed into
**  search_for_port_rep()
**
********
**
**  The node's input ports are found in the node schema's HAS_INPUT_PORTS
**  slot; the output ports are found in the node schema's HAS_OUTPUT_PORTS
**  slot.  The a pointer to an array of pointers to port_rep structures
**  for input ports is put in the panel-schema's INPUT_PORT_REPS slot, and
**  for output ports it is put in the panel-schema's OUTPUT_PORT_REPS
**  slot.
**
**  Memory is allocated for the an array of pointers to port reps that is
**  one larger than the number of ports, and also for a port_rep structure
**  for each port.  This memory should be allocated all at once so that it
**  can be freed all at once.
**
**  The array is initialized with a pointer to each successive port_rep
**  structure, terminating with a NULL pointer.
**
**  The port_rep structures are given values by iterating over the ports
**  and:
**
**      filling the port member with a pointer to the port schema
**
**      filling the menu_item member with a port-type menu item
**  created by calling create_port_type_menu_item() with the value of the
**  port's NAME slot and either the value of the port's PORT_DATA_TYPE (if
**  it is a string) or use that value to find the data-type schema and use
**  it's NAME_OF_DATA_TYPE slot.
**
*/

void construct_initial_port_reps (panel_schema, node, input)
art_symbol panel_schema;
art_symbol node;
boolean input;
{}


/*
```

```
**      construct_initial_connector_reps (art_symbol panel_schema)
**
**  This function creates the support connector_rep structures for
**  Connector Details Panels.  The necessary port_rep structures should be
**  made first with construct_initial_port_reps()
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel.
**
**
********
**
**  Any previous values in the CONNECTOR_REPS and CONNECTOR_REPS_SIZE
**  slots are reused.  If none exists, a new array is allocated with
**  some size (ed: specify constant value later) and its contents are made
**  NULL.
**
**  The connector group that the panel is for is found in the
**  CURRENTLY_DISPLAYED_CONNECTOR_GROUP slot.  The HAS_CONNECTORS slot in
**  the connector-group schema contains the list of connector objects.
**  This list is iterated over and the connector_rep structures are filled
**  in by add_new_connector_rep().  The connector menu items are
**  constructed with make_connector_menu_item().
**
**  The connector schema's SOURCE_PORT slot contains a pointer to the
**  source port schema of the connector and the DESTINATION_PORT slot
**  contains a pointer to the destination port schema.
**
**  If the panel is a Node To panel or Graph Port To panel then the source
**  argument to make_connector_menu_item() is found by calling
**  search_for_port_rep_by_port() with the port schema and using it's
**  menu_item member.  If the panel is a To Node or To Graph Port panel
**  then the destination argument is found similarily.
**
**  On Constant To panels, one must use the constant value for the source
**  argument to make_connector_menu_item().  The constant schema is
**  pointed to by the connector schema's SOURCE_PORT slot, and the value
**  is stored in its VALUE slot.
**
*/


construct_initial_connector_reps (panel_schema)
art_symbol panel_schema;




/*
**      connector_rep *add_new_connector_rep (art_symbol panel_schema,
**                                            char *connector_menu_item,
**                                            art_symbol connector,
```

```
**                                                  char *notes)
**
**  This function makes a new connector_rep structure for a new
**  connection.  It returns a pointer to the new connector_rep structure.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      connector_menu_item is a connector menu item string
**
**      connector is the corresponding connector schema
**
**      notes is the notes for the connector
**
**      the return value is a pointer to the connector rep structure
**              made.
**
********
**
**  If connector is NULL, then the connector_rep is for a newly defined
**  connector.  This is the case when this function is called from
**  CD_Select_Connect().
**
**  If connector is non-NULL, then the connector_rep is for an already
**  existing connector.  This is the case when it is called by
**  construct_initial_reps().
**
**
**  The array of connector_rep pointers is stored in the panel_schema's
**  CONNECTOR_REPS slot and the current size of that array is stored in
**  the CONNECTOR_REPS_SIZE slot.  This array is scanned for a NULL
**  pointer.  If no NULL pointer is found, then the array is enlarged as
**  follows:
**
**      1. Allocate memory a larger array of pointers.  Change the
**  CONNECTOR_REPS_SIZE slot.  The array is size increased by some
**  constant value (ed: specify constant value later).
**
**      2. Copy the old array of pointers.
**
**      3. Free up the old array of pointers.
**
**
**  A new connector_rep structure is allocated and fields are filled in as
**  follows:
**
**      1. The menu_item and notes members are set to the passed
**  connector_menu_item and a copy of notes.
**
**      2. The connector member is given the passed connector value.
**  The associated member is set to TRUE.
**
**  If this function fails (only possible if memory cannot be allocated),
**  it will return NULL.
```

```
**
**
**  COMMENTS:
**
**      - Failure is so unexpected, that maybe it should raise a warning.
**
**      - May want to break up enlarge array functionality into another
**      function.
**
*/

connector_rep *add_new_connector_rep (panel_schema,
                                      connector_menu_item, connector, notes)
art_symbol panel_schema;
char *connector_menu_item;
art_symbol connector;
char *notes;
{}



/*
**      void free_up_reps (art_symbol panel_schema);
**
**  This function frees up memory used by the panel_schema's connector_rep
**  and port_rep structures, prior to dismissing the panel.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**
**  If the panel is not a Constant To panel, a pointer to an array of
**  pointers to port_reps are found in the panel schema's INPUT_PORT_REPS
**  slot.  This memory is freed here.  All panlels have an
**  OUTPUT_PORT_REPS slot which points to memory which must be similarily
**  freed.
**
**  This function iterates over the array of pointers to connector_rep
**  structures found in the panel_schema's CONNECTOR_REPS slots.  The size
**  of the array is stored in the CONNECTOR_REPS_SIZE slot.
**
**  **  All connector_rep structs pointed to by the CONNECTOR_REPS array have
**  their menu_item and notes member strings freed.  Then the memory used
**  by the connector_rep structure is then freed and the pointers in the
**  array are set to NULL.
**
*/

void free_up_reps (panel_schema)
art_symbol panel_schema;
{}
```

```
/*
**      void create_and_destroy_changed_connectors (art_symbol panel_schema);
**
**  This function creates a connector schema for each newly defined
**  connector, modifies connectors with changed definitions, and destroys
**  newly undefined connectors.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**  It iterates over the array of pointers to connector_reps found in the
**  CONECTOR_REPS slot of the passed panel_schema.  The size of that array
**  is found in the CONNECTOR_REPS_SIZE slot.
**
**  For each connector rep structure pointed to:
**
**      - If the value of connector member is non-NULL and associated
**  is FALSE, then the corresponding connector schema must be destroyed.
**  This requires that:
**
**              1. The connector group schema no longer reference it.
**
**              2. The memory used by the menu_item and notes fields
**  is freed.  The memory used by the connector_rep is also freed.  The
**  pointer to the connector_rep structure is set to NULL.
**
**              3. The connector schema be deleted
**
**
**      - If the value of the connector member is non-NULL and
**  associated is TRUE, then the corresponding connector schema may need
**  to have their NOTES slots changed.  The notes member contains current
**  notes.
**
**
**      - If the value of connector member is NULL and associated is
**  TRUE, then a new connector object must be created.  This requires
**  that:
**
**              1. A new connector schema is made with NOTES and
**  BELONGS_TO_CONNECTOR_GROUP slots filled in.
**
**              2. The connector group must reference this new
**  connector object in its HAS_CONNECTORS slot.
**
**              3. The connector field in the connector_rep must point
**  the new connector object.
**
*/
```

```
void create_and_destroy_changed_connectors (panel_schema)
art_symbol panel_schema;
{}



/*
**      port_rep *search_for_port_rep_by_menu_item (art_symbol panel_schema,
**                                                  char *port_type_menu_item,
**                                                  boolean input);
**
**      port_rep *search_for_port_rep_by_port (art_symbol panel_schema,
**                                             art_symbol port,
**                                             boolean input);
**
**  These functions searche for the port_rep structure that corresponds to
**  a port-type menu item or port schema, given the port direction.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
**      port_type_menu_item is the port-type menu item to be searched
**              for.  port-type menu item format is discussed in
**              construct_port_type_menu_item()
**
**      port is the port schema
**
**      input is TRUE if the port is an input port, FALSE otherwise
**
********
**
**  If input is TRUE, the panel schema's INPUT_PORT_REPS slot is used for
**  the search.  Otherwise, the panel schema's OUTPUT_PORT_REPS slot is
**  used.
**
**  The panel schema's [INPUT|OUTPUT]_PORT_REPS slot contains a pointer to
**  a NULL terminated array of pointers to port_rep structures.
**
**  search_for_port_rep_by_menu_item() searches these port_rep structures
**  for one that has its menu_item member that strcmp() with the passed
**  port-type menu item.
**
**  search_for_port_rep_by_port compares the port_rep's port member with
**  the passed port (ed: with a_eq or = ?).
**
**
**  COMMENTS:
**
**  - It is an internal error to for the panel schema not to have the
**  proper slot, or for there to be no value in this slot.  Since it is
**  probably caused by a logic flaw we will want fixed, a warning message
**  should be generated.
```

```
**
*/

port_rep *search_for_port_rep_by_menu_item (panel_schema,
                                            port_type_menu_item, input);
art_symbol panel_schema;
char *port_type_menu_item;
boolean input;
{}

port_rep *search_for_port_rep_by_port (panel_schema, port, input)
art_symbol panel_schema;
art_symbol port;
boolean input;
{}



/* populate.c */


/*
**      art_symbol populate_new_connector_details_panel
**              (art_symbol connector_group);
**
**  This function populates a new Connector Details panel for the passed
**  source and destination objects.  The type of panel is determined by
**  the types of the source and destination objects.  The panel schema
**  associated with the new panel is returned.
**
**  Where:
**
**      connector_group is the connector-group schema that the
**              Connector Details Panel is for.
**
**      the returned value is the schema that corresponds to the new
**              Connector Details Panel
**
********
**
**  The panel schema is found with find_screen_schema() (?).
**
**  After putting the connector_group in the
**  CURRENTLY_DISPLAYED_CONNECTOR_GROUP slot, the connector_reps are
**  constructed with construct_initial_reps().
**
**  The connectors textlist is populated with
**  populate_connectors_textlist().
**
**  No item is selected in the connectors textlist.  The Notes field is
**  also cleared.
**
**  MAS: More needs to be done, but I'm not sure about how and I'll think about
**  it later:
**
```

```
**      1. Sources textlist and Destinations textlist for Node To and
**  To Node panels.  Titles of textlists too.
**
**      2. The Constant source, the Graph Port Name fields must be populated
**

MAS:
        here is the time for port_rep structures.  Also look for
CD_Select_Source(), etc.

*/

art_symbol populate_new_connector_details_panel (connector_group)
art_symbol connector_group;
{}



/*
**      void populate_connectors_textlist (art_symbol panel_schema);
**
**  This function populates the connectors textlist.
**
**  Where:
**
**      panel_schema is the schema corresponding to the Connector
**              Details Panel
**
********
**
**  A pointer to an array of pointers to the connector_rep structures is
**  found in the panel schema's CONNECTOR_REPS slot and the size of this
**  array is found in the CONNECTOR_REPS_SIZE slot.  This array is
**  iterated over and connector_rep structs pointed to have their
**  menu_item member as one of the items populated in the connectors
**  textlist, if their associated member is TRUE.
**
**
**  COMMENTS:
**      This function is so simple, does it need to be?
**
*/

void populate_connectors_textlist (panel_schema)
art_symbol panel_schema;
{}



/*
**      art_symbol populate_new_node_details_panel (art_symbol node);
**
**  This function creates, populates, and pops up a Node Details panel.
**
```

```
**  Where:
**
**      node is the node schema that the panel is for
**
**      the return value is the schema that corresponds to the Node
**              Details Panel.
**
********
**
**  Must populate the following fields:
**
**      - Name text field (from node's NAME slot)
**
**      - Type static text (using esl_object_type_string())
**
**      - On Graph static text (from node's ON_GRAPH slot)
**
**      - Input Ports and Output Ports textlists (from node's
**              HAS_INPUT_PORTS and HAS_OUTPUT_PORTS slots and
**              make_port_status_menu_item())
**
**  The memory allocated by make_port_status_menu_item() is freed here.
**
*/

art_symbol populate_new_node_details_panel (node)
art_symbol node;
{}


/*
**      art_symbol populate_new_component_details_panel
**              (art_symbol component);

**  This function creates, populates, and pops up a Component Details
**  panel.
**
**  Where:
**
**      component is the component schema the panel is for
**
**      the return value is the schema that corresponds to the
**              Component Details Panel
**
********
**
**  Must populate the following fields:
**
**      - Name static text (from component's NAME slot)
**
**      - Type static text (using esl_object_type_string())
**
**      - Input Ports and Output Ports textlists (from component's
**              HAS_INPUT_PORTS and HAS_OUTPUT_PORTS slots and also
**              using make_port_status_menu_item())
**
```

```
**      - Implementation static text fields:
**              Type (using esl_implementation_type_string())
**
**              Package name (either the constant EMPTY_PACKAGE_NAME or
**                      from the components HAS_IMPLEMENTATION slot
**                      and the implementation schema's
**                      NAME_OF_PACKAGE slot, if it is an instance of
**                      PACKAGE_IMPLEMENTATION)
**
**              Spec and Body filename (if implementation schema is
**                      instance of INLINE_IMPLEMENTATION, then the
**                      constant string NO_FILE_FILENAME.  If
**                      implementation schema is instance of
**                      SEPARATELY_COMPILED_PROCEDURE then it is the
**                      SOURCE_FILE_NAME slot for the spec and the
**                      NO_FILE_FILENAME constant for the body.  If
**                      the implementation schema is an instance of
**                      PACKAGE_IMPLEMENTATION, then the slots
**                      PACKAGE_SPEC_FILE_NAME and
**                      PACKAGE_BODY_FILE_NAME are used.)
**
**      - Node instances textlist (from the component's
**              CORRESPONDS_TO_NODES slot and the respective node's
**              NAME slot)
**
**  The memory allocated by make_port_status_menu_item() is freed here.
**
*/

art_symbol populate_new_component_details_panel (component)
art_symbol component;
{}



/*
**      art_symbol populate_new_graph_port_details_panel (art_symbol graph,
**                                                        boolean input);
**
**  This function creates, populates, and pops up a Graph Port Details
**  panel.
**
**  Where:
**
**      graph is the graph schema that the panel is for
**
**      input is TRUE if the panel is for graph _input_ ports, FALSE
**              it the pane is for graph _output_ ports.
**
**      the return value is the schema that corresponds to the Graph
**              Port Details Panel.
**
********
**
**  Must populate the following fields:
**
```

```
**      - Name static text (from graph's NAME slot)
**
**      - Graph Ports textlist
**        Title is either GRAPH_INPUT_PORTS_STRING or
**        GRAPH_OUTPUT_PORTS_STRING constants.
**        Items made with make_port_status_menu_item()
**
**  The memory allocated by make_port_status_menu_item() is freed here.
**
*/

art_symbol populate_new_graph_port_details_panel (graph, input)
art_symbol graph;
boolean input;
{}



/*
**      art_symbol populate_new_node_notes_panel (art_symbol node);
**
**  This function creates, populates, and pops up a Node Notes panel.
**
**  Where:
**
**      node is the node schema that the panel is for
**
**      the return value is the schema that corresponds to the Node
**              Notes Panel.
**
********
**
**  Must populate the following fields:
**
**      - Name static text (from node's NAME slot)
**
**      - Type static text (using esl_object_type_string())
**
**      - Component notes textdisp (from node's uses-subprogram slot or a
**      constant string and the subprogram's NOTES slot)
**
**      - Node notes pageedit (from node's NOTES slot)
**
*/

art_symbol populate_new_node_notes_panel (node)
art_symbol node;
{}



/*
**      art_symbol populate_new_component_notes_panel (art_symbol component);
**
**  This function creates, populates, and pops up a Component Notes panel.
**
```

```
**  Where:
**
**      component is the componenent schema that the panel is for
**
**      the return value is the schema that corresponds to the Component
**              Notes Panel.
**
********
**
**  Must populate the following fields:
**
**      - Name static text (from component's NAME slot)
**
**      - Type static text (using esl_object_type_string())
**
**      - Component Notes textdisp field (from component's NOTES slot)
**
*/

art_symbol populate_new_component_notes_panel (component)
art_symbol component;
{}


/* MAS: explain the return value... must be allocated */

/*
**      char *select_constant_value_modal_panel
**              (char *data_type,
**               art_symbol parent_panel);
**
**  This function creates, populates, and pops up a modal Select Constant
**  Value panel.  The panel has exclusive focus.  Control is returned to
**  the caller only after the user selects either the Ok or Cancel button.
**
**  Where:
**
**      data_type is the string for the data type that the panel is
**              for.  Will probably be changed to an art_symbol for
**              the data-type schema.
**
**      the return value is the constant value string the user
**              selected from the Defined Values textlist if Ok is
**              selected, or NULL if Cancel is selected.
**
********
**
**  Must populate the following fields:
**
**      - Data type - (from passed argument)
**
**      - Defined values (from the data type schema's DEFINED_VALUES slot)
**
**
**  COMMENTS:
```

```
**
**      Mapping between data type name and data type schema must be found.
**
*/


char *select_constant_value_modal_panel (data_type, parent_panel)
char *data_type;
art_symbol parent_panel;
{}



/* misc.c */

/*
**      boolean propogate_constraints();
**
**  This function propogates constraints.
**
**  Where:
**
**      the return value is FALSE if constraints are violated, TRUE
**              otherwise.
**
********
**
**  EWR knows how to write this function.
*/

boolean propogate_constraints()
{}



/*
**      boolean type_check_constant (char *constant_value,
**                                   char *data_type);
**
**  This function checks to see if a constant value is legal for a
**  particular data type.
**
**  Where:
**
**      constant_value is the string of the constant value to be checked
**
**      data_type is the string of the data type name.  This will
**              probably be changed to be a data type schema.
**
**      the return value is TRUE if constant_value is valid for data_type.
**
********
**
**  First get data type schema (how?).
**
```

```
**  If the data type is BOOLEAN, INTEGER, FLOAT, or STRING, then check
**  first with legal_boolean_constant_p(), legal_integer_constant_p(),
**  legal_float_constant_p(), or legal_string_constant_p().
**
**  If that test fails then check if the value is on the data type
**  schema's DEFINED_VALUES slot.
**
**  If that test fails then call the function in the data type schema's
**  TEST_FUNCTION slot.
**
**  If all tests fail, return FALSE.
**
**
**  COMMENTS:
**
**      - Mapping between data type string and data type schema must
**  be found.
*/

boolean type_check_constant (char *constant_value,
                             char *data_type)
char *constant_value, *data_type;
{}



/*
**      boolean type_check_graph_port (char *graph_port_name,
**                                     char *data_type);
**
**  This function checks to see if a particular Graph Port is legal for a
**  to be connected to a port with some data type.
**
**  Where:
**
**      graph_port_name is the name of the graph port to be connected.
**              This will probably be changed to be a graph_port schema.
**
**      data_type is the string of the data type name.  This will
**              probably be changed to be a data type schema.
**
**      the return value is TRUE if the graph port may be connected to
**              a port with the passed data type.
**
********
**
**  First find the graph port schema and the data type schema.
**
**  If the graph port schema doesn't exist, then it is new, and therefore
**  Ok.
**
**  Otherwise, the graph port schema's PORT_DATA_TYPE slot points to the
**  port's data type schema.  The data type's NAME_OF_DATA_TYPE slot
**  contains the data type name.  It must compare with the passed data
**  type.
**
```

```
**
**  COMMENTS:
**
**      - Mapping to graph_port schema and data type schema is
**  necessary.
**
*/

boolean type_check_graph_port (graph_port_name, data_type)
char *graph_port_name, data_type;
{}



/*
**      boolean legal_boolean_constant_p (char *boolean_constant);
**      boolean legal_integer_constant_p (char *integer_constant);
**      boolean legal_float_constant_p (char *float_constant);
**      boolean legal_string_constant_p (char *string_constant);
**
**
**  Thes functions check to see if a value is of a particular Ada
**  predefined type.
**
**  Where
**
**      boolean_constant, integer_constant, float_constant,
**              string_constant are strings with the value to be checked
**
**      the return value is TRUE if the argument is of the type being
**              checked, FALSE otherwise.
**
********
**
**  Legal boolean values are "TRUE", "FALSE", or any casification of those
**  two.  Convert the passed string to uppercase and compare with "TRUE"
**  and "FALSE".
**
**
**  Legal integers are: digit [digit]* (ie: "123")
**
**
**  Legal floats are: digit [digit]* "." [digit]* ["E" ["+" | "-"] [digit]*]
**
**
**  Legal strings are: "\"" [char]* "\""
**
**
**  COMMENTS:
**
**      - Another legal integer is: integer "#" integer++ "#".  The
**  first integer is the base.  The second integer has broader
**  interpretations. (ie: 16#12FA#)
**
*/
```

```
boolean legal_boolean_constant_p (boolean_constant)
char *boolean_constant;
{}


boolean legal_integer_constant_p (integer_constant)
char *integer_constant;
{}


boolean legal_float_constant_p (float_constant)
char *float_constant;
{}


boolean legal_string_constant_p (string_constant)
char *string_constant;
{}
```